IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

MICROSOFT CORPORATION
Petitioner,

v.

EDGE NETWORKING SYSTEMS, LLC,
Patent Owner.

U.S. Patent No. 10,893,095
Issue Date: January 12, 2021
Title: DISTRIBUTED SOFTWARE DEFINED NETWORKING

Inter Partes Review No.: Unassigned

DECLARATION OF JUNE ANN MUNFORD
ON BEHALF OF PETITIONER

Mail Stop "PATENT BOARD"
Patent Trial and Appeal Board
U.S. Patent and Trademark Office
P.O. Box 1450
Alexandria, VA 22313-1450

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0001

TABLE OF CONTENTS

I. Introduction 1

A. Qualifications and Professional Experience 1

B. Scope of This Declaration 6

C. Person of Ordinary Skill in the Art 8

II. AUTHENTICITY AND PUBLIC AVAILABILITY OPINIONS 9

A. Authenticity and Public Availability of *Dependability as a Cloud Service - A Modular Approach* (**Exhibit 001**) 9

1. Authentication 9

2. Public Availability 10

3. Summary of My Opinions on *Dependability as a Cloud Service - A Modular Approach* (**Exhibit 001**) 17

B. Authenticity and Public Availability of *OSGi in Depth* (**Exhibit 002**) 19

1. Authentication 19

2. Public Availability 23

3. Summary of My Opinions on *OSGi in Depth* (**Exhibit 002**) 27

C. Authenticity and Public Availability of *OSGi in Action: Creating Modular Applications in Java* (**Exhibit 003**) 28

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0002

1. Authentication 28

2. Public Availability.....33

3. Summary of my Opinions on *OSGi in Action: Creating Modular Applications in Java* (**Exhibit 003**) 38

III. CONCLUSION 39

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0003

I. INTRODUCTION

I, June Ann Munford, do hereby declare as follows:

1. My name is June Ann Munford. I am over the age of 18, have personal knowledge of the facts set forth herein, and am competent to testify to the same.

2. I have been retained as an independent expert consultant on behalf of Microsoft Corporation ("Petitioner") for *inter partes* review ("IPR") proceedings before the United States Patent and Trademark Office ("PTO") regarding the authenticity and public availability of certain prior art references in relation to U.S. Patent Nos. 10,686,871, 10,893,095 and 11,695,823.

3. I am being compensated for my work in this matter at my customary hourly rate. I am also being reimbursed for any reasonable expenses associated with my work and testimony in this investigation. My compensation is not contingent on the results of my study, the substance of my opinions, or the outcome of this matter.

A. Qualifications and Professional Experience

4. My complete qualifications and professional experience are described in my academic curriculum vitae (**Appendix CV**). The following is a brief summary of my relevant qualifications and professional experience.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0004

5. I earned a Master of Library and Information Science (MLIS) from the University of Wisconsin-Milwaukee in 2009. I have over ten years of experience in the library/information science field. Beginning in 2004, I have served in various positions in the public library sector including Assistant Librarian, Youth Services Librarian and Library Director.

6. During my career in the library profession, I have been responsible for materials acquisition for multiple libraries. In that position, I have cataloged, purchased and processed incoming library works. That includes purchasing materials directly from vendors, recording publishing data from the material in question, creating detailed material records for library catalogs and physically preparing that material for circulation. In addition to my experience in acquisitions, I was also responsible for analyzing large collections of library materials, tailoring library records for optimal catalog search performance and creating lending agreements between libraries during my time as a Library Director.

7. I am fully familiar with the catalog record creation process in the library sector. In preparing a work for public availability, a library catalog record describing that work would be created. These records are typically written in Machine Readable Catalog (herein referred to as "MARC") code, which became the national standard for sharing bibliographic data in the United States by 1971

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0005

and the international standard by 1973. MARC is the primary communications protocol for the transfer and storage of bibliographic metadata in libraries. Experts in my field reasonably rely upon MARC records when forming their opinions.

8. A MARC record consists of several fields, each of which contains specific data about the work, such as a physical description of the work, metadata from the work's publisher, and date of library acquisition. Each field is identified by a standardized, unique, three-digit code corresponding to the type of data that follows. **Appendix B** is a true and correct copy of Parts VII to X of "Understanding MARC Bibliographic: Machine-Readable Cataloging" (http://www.loc.gov/marc/umb/um07to10.html) from the Library of Congress that explains commonly used MARC fields. Each field of a MARC record is identified by a standardized, unique, three-digit code corresponding to the type of data that follows. For example, the 100 field of a MARC record is reserved for denoting the personal author of a work, the 245 field is reserved for denoting the title, the 260 field is reserved for denoting the publisher information, field 300 is reserved for denoting the physical description of a publication, field 650 is reserved for denoting topical subjects, and the 008 field is reserved for denoting the date of creation of the library record itself. As this typically occurs during the process of preparing works for public access, it is my experience that a work's MARC record indicates the date of a work's public availability.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0006

9. Typically, in creating a MARC record, a librarian would gather various bits of metadata such as title of the work, publisher and subject headings among others and assign each value to a relevant numerical field. For example, a work's physical description is tracked in field 300 while title/attribution is tracked in field 245.

10. Librarians create MARC records for works that the libraries acquire, including books, serials, motion pictures, and publications in other formats. MARC records are representations of materials held by libraries and are critical for the discovery and access to those materials. Most libraries make a newly cataloged work available to the public soon after they create the MARC record, with the average processing cycle requiring, based on my experience, somewhere between 1-3 weeks for a given periodical material, and 2-3 months for a given book material, to be made available to the public. The 008 field of the MARC record is reserved for denoting the creation of the library record itself. As this is the only date reflecting the inclusion of said work within the library's collection, it is my experience that an item's 008 field is an accurate indicator of the date of a work's public availability.

11. I am familiar with many online catalogs, databases, and search engines. In preparing this declaration, I used the following authoritative information systems to search for records:

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0007

- ACM Digital Library (https://dl.acm.org/)
- Ecosia (https://www.ecosia.org/)
- Google Scholar (https://scholar.google.com/)
- HAL Theses (https://theses.hal.science/)
- International Journal of Computer Applications (http://ijcaonline.org)
- IEEE Xplore Digital Library (https://ieeexplore.ieee.org/Xplore/home.jsp)
- Internet Archive (https://archive.org/)
- Manning Publications Co. (https://www.manning.com/)
- OCLC Connexion (https://www.oclc.org/en/connexion.html)
- OCLC FirstSearch (https://firstsearch.oclc.org/)
- Online catalog of Library of Congress (https://catalog.loc.gov/)
- Online catalog of the Massachusetts Institute of Technology Library (https://libraries.mit.edu/)
- Online catalog of the Rochester Institute of Technology (https://www.rit.edu/library/)
- Online catalog of the University of Chicago Library (https://www.lib.uchicago.edu/)
- Online catalog of the University of Washington Library (https://lib.uw.edu/)
- Online catalog of the Villanova University Library (https://library.villanova.edu/)

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0008

- Public catalog of the US Copyright Office (https://cocatalog.loc.gov/cgi-bin/Pwebrecon.cgi?DB=local&PAGE=First)
- Semantic Scholar (https://www.semanticscholar.org/)
- VDE Verlag (https://www.vde-verlag.de/english.html)
- WorldCat (https://search.worldcat.org/)

B. Scope of This Declaration

12. I have been asked to offer an opinion on the authenticity and public availability date of the following documents:

1) "Dependability as a Cloud Service - A Modular Approach" by Jan S. Rellermeyer, et al as published in *Proceedings of the International Conference on Dependable Systems and Networks, 2012*, ©2012, **Exhibit 001** ("*Dependability as a Cloud Service - A Modular Approach*");
2) *OSGi in Depth* by J.G. Alexandre de Castro Alves, Manning Publications Company, ©2012, **Exhibit 002**, ("*OSGi in Depth*"); and
3) *OSGi in Action: Creating Modular Applications in Java* by Richard S. Hall, et al., Manning Publications Company, ©2011, **Exhibit 003**, ("*OSGi in Action: Creating Modular Applications in Java*").

13. I am rendering my expert opinion on the authenticity of the documents referenced herein and on whether they were published. I will provide my opinion on when and how these documents were publicly disseminated or

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0009

otherwise made publicly available to the extent that persons interested and ordinarily skilled in the subject matter or art, exercising reasonable diligence, could have located and retrieved these documents.

14. I am informed by counsel that an item is considered authentic if there is sufficient evidence to support a finding that the item is what it is claimed to be. I am also informed that authenticity can be established based on the contents of the documents themselves, such as the appearance, content, substance, internal patterns, protocols followed or other distinctive characteristics of the item.

15. I am informed by counsel that a given reference is "published" if it is publicly accessible upon a satisfactory showing that such a document has been disseminated or otherwise made publicly available to the extent that persons interested and ordinarily skilled in the subject matter or art could locate it exercising reasonable diligence and obtain the document. I have also been informed by counsel that materials available in a library constitute "printed publications" if they are cataloged and indexed according to general library practices and protocols that make the references available and accessible to members of the interested public.

16. I have been informed by counsel and understand that U.S. Patent Nos. 10,686,871, 10,893,095 and 11,695,823 each claim an earliest priority date of June

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0010

13, 2013. I have assessed public accessibility of the three (3) documents referenced above as of this claimed priority date.

C. Person of Ordinary Skill in the Art

17. I have been informed by counsel and understand that a "person of ordinary skill in the art" ("POSITA") is a hypothetical person who is presumed to be familiar with the relevant field and aware of all pertinent prior art at the time of the invention. I have also been informed by counsel and understand that this hypothetical person is also a person of ordinary creativity, capable of understanding the scientific principles and literature applicable to the pertinent field.

18. I am informed by counsel and understand that a POSITA in the field of U.S. Patent Nos. 10,686,871, 10,893,095 and 11,695,823, as of their June 13, 2013 priority date, would have would have had a bachelor's degree in computer science, computer engineering, or equivalent degree, and approximately three years of experience working in the computer science or engineering field. Additional experience might substitute for less education and vice versa. I am also informed by counsel and understand that a POSITA in June 2013 would have been knowledgeable about the design and management of networked systems and virtualization technologies, and familiar with operating/distributed systems and security and privacy techniques.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0011

19. It is my opinion that such a person would have been actively engaged in learning about the field, possibly through formal instruction and bibliographic resources. By at least June 2013, such a person would have had access to a vast array of print and electronic resources, including at least those resources referenced below.

II. AUTHENTICITY AND PUBLIC AVAILABILITY OPINIONS

A. Authenticity and Public Availability of *Dependability as a Cloud Service - A Modular Approach* (Exhibit 001)

1. Authentication

20. **Exhibit 001** of my Declaration is a true and correct copy of "Dependability as a Cloud Service - A Modular Approach" as published in *Proceedings of the International Conference on Dependable Systems and Networks, 2012* as held by the Rochester Institute of Technology library's subscription to the IEEE Xplore service. I secured this PDF myself from the library's collection.

21. I have been informed by counsel and understand that Exhibit 1011 submitted by Petitioner in the IPR proceedings is the same true and correct copy that I secured from the Rochester Institute of Technology library's collection and attach as **Exhibit 001** to my Declaration.

22. Page 1 of **Exhibit 001** shows the title, authors and abstract, includes an ISBN (International Standard Book Number) of 978-1-4673-2266-9, and has a

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0012

"2012" copyright with the IEEE as the copyright holder. **Exhibit 001** shows *Dependability as a Cloud Service - A Modular Approach* has a total of six (6) pages, two (2) figures, one (1) table, and nineteen (19) references.

23. This Rochester Institute of Technology copy of *Dependability as a Cloud Service - A Modular Approach* (**Exhibit 001**) is in a condition that creates no suspicion about its authenticity. Specifically, this copy is not missing any intermediate pages, the text on each page appears to flow seamlessly from one page to the next, and there are no visible alterations to the document. Moreover, this copy was found in the custody of a library, a place where, if authentic, it would likely be found. I therefore see no reason to question the authenticity of this copy of *Dependability as a Cloud Service - A Modular Approach* (**Exhibit 001**).

2. Public Availability

24. Attached hereto as **Appendix RELLERMEYER03** is a true and correct copy of a screen capture of the IEEE Xplore metadata record for *"Dependability as a Cloud Service - A Modular Approach"*. I personally located, identified, and obtained these screen captures myself from https://ieeexplore.ieee.org/document/6264688?partnum=6264688&searchProductType=IEEE Conferences. This is the type of record that experts in my field would rely upon when forming their opinions.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0013

25. PDF Pages 1 to 2 of **Appendix RELLERMEYER03** are the *IEEE Xplore* metadata record that shows the title, authors (Jan Rellermeyer and Saurabh Bagchi), publisher (IEEE), digital object identifier, abstract and ISBNs (electronic, print, online) of *"Dependability as a Cloud Service - A Modular Approach"*. It also shows *"Dependability as a Cloud Service - A Modular Approach"* appears in the *Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)*, the conference dates (June 25-28, 2012), conference location (Boston, MA), and that the article publication date is "13 August 2012." PDF Pages 2-10 of Appendix RELLERMEYER03 are the "Contents" of the *IEEE Xplore* metadata record of *"Dependability as a Cloud Service - A Modular Approach"* that shows the HTML version of the text and figures of this article.

26. PDF Pages 11 to 26 of **Appendix RELLERMEYER03** are the *IEEE Xplore* metadata record for the *Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)* that shows the conference chairs, publisher (which is IEEE), conference date, and digital object identifier of the proceedings, and also lists the articles published in the proceedings. PDF Pages 22-23 shows an entry for *"Dependability as a Cloud Service - A Modular Approach"* that includes a link to this article, and again shows the authors of this article.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0014

27. PDF Pages 27 to 32 of **Appendix RELLERMEYER03** are a true and correct copy of the work as downloaded as a PDF from this *IEEE Xplore* page. I have closely compared **Exhibit 001** (Rochester Institute of Technology copy of *Dependability as a Cloud Service - A Modular Approach*) with the copy purchased from IEEE (PDF pages 27 to 32 of **Appendix RELLERMEYER03**) and concluded that these two copies have the same content.

28. The IEEE record made *"Dependability as a Cloud Service - A Modular Approach"* discoverable in the IEEE Xplore system from August 13, 2012 on. As of that date, interested persons exercising reasonable diligence would have been able to locate IEEE record for *Dependability as a Cloud Service - A Modular Approach* and use the link in this record to purchase the article from IEEE or obtain it from an institution that subscribed to IEEE Xplore (*e.g.*, the Rochester Institute of Technology library).

29. Actual dissemination of a publication (such as *Dependability as a Cloud Service - A Modular Approach*) to interested persons exercising reasonable diligence is reflected by other publications, especially those of other authors, that make reference to it.

30. Attached hereto as **Appendix RELLERMEYER06** is (i) a screen capture, from the ACM Digital Library website, of bibliographic data for a research article entitled "Component-based scalability for cloud applications",

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0015

authored by Steffen Kachele and Franz Hauck, published in *CloudDP '13: Proceedings of the 3rd International Workshop on Cloud Data and Platforms*, pages 19-24, April 14, 2013, as well as (ii) a true and complete copy of the research article of the same name that I retrieved from the same website. I secured these screen captures myself from https://dl.acm.org/doi/abs/10.1145/2460756.2460760. These are the type of records that experts in my field would reasonably rely upon when forming their opinion.

31. **Appendix RELLERMEYER06** indicates this research article entitled "Component-based scalability for cloud applications" was published on April 14, 2013. On PDF page 9 of **Appendix RELLERMEYER06**, I note that this research article references: "[16] J. Rellermeyer and S. Bagchi. Dependability as a cloud service - a modular approach. In IEEE/IFIP 42nd Int. Conf. on Dep. Sys. and Netw. Workshops (DSN-W), pages 1–6, 2012", which is the *Dependability as a Cloud Service - A Modular Approach* article. I also note that the named authors of this research article entitled "Component-based scalability for cloud applications" are distinct from the named authors of the *Dependability as a Cloud Service - A Modular Approach* article.

32. I also personally located, identified, and obtained three exemplary library records for the *Proceedings of the International Conference on Dependable*

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0016

Systems and Networks, 2012, of which (as I discussed above) *Dependability as a Cloud Service - A Modular Approach* was included therein. These are the type of records that experts in my field would reasonably rely upon when forming their opinions.

33. Attached hereto as **Appendix RELLERMEYER01** is a true and correct copy of the MARC record for *Proceedings of the International Conference on Dependable Systems and Networks, 2012* as held by the Rochester Institute of Technology library. I secured this record myself from the library's public catalog. This is the type of record that experts in my field would rely upon when forming their opinions.

34. The MARC record contained within **Appendix RELLERMEYER01** includes fields denoting, for example, the conference title, conference date and location, ISBNs of the proceedings (print (9781467322645) and online (9781467322652)), and copyright date (2012) of the *Proceedings of the International Conference on Dependable Systems and Networks, 2012*, which match such information in **Appendix RELLERMEYER03** (the publisher's (IEEE) IEEE Xplore Metadata record for this article).

35. The 008 field of the MARC record in **Appendix RELLERMEYER01** indicates the date of record creation. The 008 field of Appendix RELLERMEYER01 indicates the Rochester Institute of Technology

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0017

library first acquired these proceedings as of September 5, 2012. Considering this information, it is my determination that the *Proceedings of the International Conference on Dependable Systems and Networks, 2012*, and therefore "Dependability as a Cloud Service - A Modular Approach" which was a part of such proceedings, was made available to the public at the Rochester Institute of Technology library shortly after its initial acquisition as of September 5, 2012.

36. Attached hereto as **Appendix RELLERMEYER04** is a true and correct copy of the MARC record for the *Proceedings of the International Conference on Dependable Systems and Networks, 2012* as held by the Villanova University library. I secured this record myself from the library's public catalog. This is the type of record that experts in my field would rely upon when forming their opinions.

37. The MARC record contained within **Appendix RELLERMEYER04** includes fields denoting, for example, the conference title, conference date and location, ISBNs of the proceedings (print (9781467322645) and online (9781467322652)), and copyright date (2012) of the *Proceedings of the International Conference on Dependable Systems and Networks, 2012*, which match such information in **Appendix RELLERMEYER03** (the publisher's (IEEE) IEEE Xplore Metadata record for this article).

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0018

38. The 008 field of the MARC record in **Appendix RELLERMEYER04** indicates the date of record creation. The 008 field of **Appendix RELLERMEYER04** indicates the Villanova University library first acquired these proceedings as of September 5, 2012. Considering this information, it is my determination that the *Proceedings of the International Conference on Dependable Systems and Networks, 2012*, and therefore "Dependability as a Cloud Service - A Modular Approach" which was a part of such proceedings, was made available to the public at the Villanova University library shortly after its initial acquisition as of September 5, 2012.

39. Attached hereto as **Appendix RELLERMEYER05** is a true and correct copy of the MARC record for the *Proceedings of the International Conference on Dependable Systems and Networks, 2012* as held by the University of Washington library. I secured this record myself from the library's public catalog. This is the type of record that experts in my field would rely upon when forming their opinions.

40. The MARC record contained within **Appendix RELLERMEYER05** includes fields denoting, for example, the conference title, conference date and location, ISBNs of the proceedings (print (9781467322645) and online (9781467322652)), and copyright date (2012) of the *Proceedings of the International Conference on Dependable Systems and Networks, 2012*, which

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0019

match such information in **Appendix RELLERMEYER03** (the publisher's (IEEE) IEEE Xplore Metadata record for this article)

41. The 008 field of the MARC record in **Appendix RELLERMEYER05** indicates the date of record creation. The 008 field of **Appendix RELLERMEYER05** indicates the University of Washington library first acquired these proceedings as of September 5, 2012. Considering this information, it is my determination that that the *Proceedings of the International Conference on Dependable Systems and Networks, 2012*, and therefore "Dependability as a Cloud Service - A Modular Approach" which was a part of such proceedings, was made available to the public at the University of Washington library shortly after its initial acquisition as of September 5, 2012.

3. Summary of My Opinions on *Dependability as a Cloud Service - A Modular Approach* (Exhibit 001)

42. My analysis of the copy of *Dependability as a Cloud Service - A Modular Approach* from the Rochester Institute of Technology library (**Exhibit 001**) informs my opinion that the content of **Exhibit 001** is authentic and is from published conference proceedings that are publicly accessible. The publisher's webpage (*see* **Appendix RELLERMEYER03**) indicates that *Dependability as a Cloud Service - A Modular Approach* was published in the *Proceedings of the International Conference on Dependable Systems and Networks, 2012* on August 13, 2012. As of that date, interested persons exercising reasonable diligence would

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0020

have been able to locate the IEEE record for *Dependability as a Cloud Service - A Modular Approach* and use the link in this record to purchase the article from IEEE or obtain it from an institution that subscribed to IEEE Xplore (*e.g.*, the Rochester Institute of Technology library). An exemplary citation record (**Appendix RELLERMEYER06**) shows that *Dependability as a Cloud Service - A Modular Approach* was actually disseminated to interested persons exercising reasonable diligence at least as early as April 2013. The MARC records in **Appendices RELLERMEYER01**, **RELLERMEYER04** and **RELLERMEYER05** respectively indicate that the Rochester Institute of Technology library, the Villanova University library, and the University of Washington library each acquired the *Proceedings of the International Conference on Dependable Systems and Networks, 2012*, of which *Dependability as a Cloud Service - A Modular Approach* was included therein, as of September 5, 2012.

43. Based on my review and analysis of these documents and records, it is my opinion that *Dependability as a Cloud Service - A Modular Approach* was cataloged and became publicly accessible no later than late September 2012 (*see, e.g.*, ¶¶10, 42 above) from any of the Rochester Institute of Technology library, the Villanova University library, and the University of Washington library, and that *Dependability as a Cloud Service - A Modular Approach* was actually

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0021

disseminated to interested persons exercising reasonable diligence at least by April 2013.

B. Authenticity and Public Availability of *OSGi in Depth* (Exhibit 002)

1. Authentication

44. **Exhibit 002** of my Declaration is a true and correct copy of select pages of *OSGi in Depth* by J.G. Alexandre de Castro Alves as held by the Library of Congress. These scans were secured from the library's collection. Researching this work, I searched the Worldcat.org website for records, and a record at https://search.worldcat.org/title/793913028 identified Library of Congress as one of the holding libraries. After verifying the holdings information in the online catalog of the Library of Congress (https://lccn.loc.gov/2011284960), I requested and obtained a scan copy of pages of *OSGi in Depth* from the Library of Congress through counsel. This library copy is presented as **Exhibit 002** in this declaration.

45. I have been informed by counsel and understand that Exhibit 1008 submitted by Petitioner in the IPR proceedings is the same true and correct copy of the select pages that were scanned from the copy secured from the Library of Congress's collection and that I attach as **Exhibit 002** to my Declaration.

46. PDF Page 1 shows the front cover, and page 35 shows the back cover of this book. The front cover shows the title, author, and publisher to which this work belongs. It also shows a call number label of "QA 76 .73 .J38 A458 2012

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0022

Copy 1." The back cover shows a Library of Congress Cataloging-in-Publication (CIP) label that shows "0 028 873 659 5" as a Library of Congress control number of this record, shows the book's 13-digit ISBN (ISBN-13) is 978-1-935182-17-7 and the book's 10-digit ISBN (ISBN-10) is 1-935182-17-X, and indicates the companion website to this book (and a "free eBook for owners of this book") is at manning.com/OSGiinDepth. PDF Page 2 is the title page that shows the same title, author and publisher information as the front cover. PDF Page 3 is the copyright page that shows a "2012" copyright date, shows a hand-written call number of "QA 76 .73 .J38 A458 2012 Copy 1" (which is the same as on the label on the front cover), and shows a Library of Congress barcode label showing "2011284960" as a Library of Congress control number of this record. This page also includes the full publisher information, and shows the ISBN-13 (that is also on the back cover). PDF Pages 4 to 34 are internal pages xiii, xiv, xviii, 2-4, 6-8, 16, 126-130, 249-250, 252-255, 261-263, and 266-269.

47. My research on *OSGi in Depth* discovered the publisher's webpage for this book at https://www.manning.com/books/osgi-in-depth from the Manning Publications website. A screen capture of the publisher webpage is presented as **Appendix ALVES07** in this declaration. Pages 1-2 of this screen capture shows *OSGi in Depth* was first published in "December 2011", shows the 13-digit ISBN

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0023

(9781935182177) of *OSGi in Depth* (which matches the copyright page and back cover of **Exhibit 002**), and also states that users "can see this entire book for free."

48. My research on *OSGi in Depth* also discovered an archived version of the publisher's webpage for this book in the Internet Archive records https://web.archive.org/web/20120207013932/http://www.manning.com/alves. These are another type of record that experts in my field would reasonably rely upon when forming their opinions. A screen capture of the archived version of the publisher webpage obtained from this Internet Archive link is presented as **Appendix ALVES08** in this declaration. Page 1 of this screen capture shows *OSGi in Depth* was first published in "December 2011", shows the 13-digit ISBN (9781935182177) of *OSGi in Depth* (which matches the copyright page and back cover of **Exhibit 002**), contains a link to purchase the printed book (and the eBook) for $75.00, and shows users can freely access certain portions of *OSGi in Depth*, including Chapter 1, which the Internet Archive contains a record of this link to Chapter 1 at https://web.archive.org/web/20130308042935/http://www.manning.com/alves/SampleChapter-1.pdf. Pages 3-23 of **Appendix ALVES08** include the PDF that is archived at this Chapter 1 link. I have closely compared pages 2-4, 6-8, 16 from the copy of Chapter 1 from the archived version of the publisher's website (**Appendix ALVES08**) with those pages in the Library of Congress copy (**Exhibit**

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0024

002), and found that these pages in the library copy are the same as their corresponding pages in the publisher copy (at the archived version of the publisher's website).

49. Internet Archive is a digital library that archives scholarly materials, cultural artifacts, and other web resources on the Internet and provides the public with free access to its collections. I have used the Internet Archive in the past and am familiar with their practice of assigning an extended URL to an archived resource in the format of http://www.archive.org/web/[Year in yyyy][Month in mm][Day in dd][Time code in hh:mm:ss]/[Archived URL] to show the point of time the archived file was captured. Because Internet Archive usually does not archive web resources the moment they appear on the Internet, and a web resource must be available first before it can be archived, the archive date of a file captured by the Internet Archive means that the archived file was available before, and definitely no later than, the archive date. **Appendix ALVES08** shows the publisher webpage for *OSGi in Depth* was archived on "20120207013932" (*i.e.*, February 7, 2012, at 1 a.m., 39 minutes, 32 seconds). This archive date means the webpage would have been publicly discoverable on the Internet no later than February 7, 2012, and users would have been able to follow the links on the webpage to purchase a copy of *OSGi in Depth*.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0025

50. **Exhibit 002** is in a condition that creates no suspicion about its authenticity. Specifically, the text on each consecutive page appears to flow seamlessly, and there are no visible alterations to the document. In addition, **Exhibit 002** is in the custody of a national library. It is therefore my opinion that **Exhibit 002** is from a published book that is authentic and publicly available.

2. Public Availability

51. Attached hereto as **Appendix ALVES01** is a true and correct copy of the MARC record for *OSGi in Depth* as held by the Library of Congress. I secured this record myself from the library's public catalog. This is the type of record that experts in my field would reasonably rely upon when forming their opinions. The MARC record contained within **Appendix ALVES01** includes fields denoting the title, author, publisher, copyright date, ISBN numbers, Library of Congress Control Number (LCCN), call number, and CIP label of *OSGi in Depth*, which match such information in **Exhibit 002**.

52. The 008 field of the MARC record in **Appendix ALVES01** indicates the date of record creation. The 008 field of **Appendix ALVES01** indicates the Library of Congress first acquired *OSGi in Depth* as of February 24, 2012. Considering this information, it is my determination that *OSGi in Depth* was made discoverable in the online catalog of the Library of Congress shortly after its initial acquisition as of February 24, 2012.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0026

53. Attached hereto as **Appendix ALVES02** is a true and correct copy of the MARC record for *OSGi in Depth* as held by the Rochester Institute of Technology library. I secured this record myself from the library's public catalog. The MARC record contained within **Appendix ALVES02** includes fields denoting the title, author, publisher, copyright date, and ISBN numbers of *OSGi in Depth*, which match such information in **Exhibit 002**.

54. The 008 field of the MARC record in **Appendix ALVES02** indicates the date of record creation. The 008 field of **Appendix ALVES02** indicates the Rochester Institute of Technology library first acquired *OSGi in Depth* as of December 23, 2011. Considering this information, it is my determination that *OSGi in Depth* was made available to the public at the Rochester Institute of Technology library shortly after its initial acquisition as of December 23, 2011.

55. Attached hereto as **Appendix ALVES03** is a true and correct copy of the MARC record for *OSGi in Depth* as held by the Massachusetts Institute of Technology library. I secured this record myself from the library's public catalog. The MARC record contained within **Appendix ALVES03** includes fields denoting the title, author, publisher, copyright date, and ISBN numbers of *OSGi in Depth*, which match such information in **Exhibit 002**.

56. The 008 field of the MARC record in **Appendix ALVES03** indicates the date of record creation. The 008 field of **Appendix ALVES03** indicates the

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0027

Massachusetts Institute of Technology library first acquired *OSGi in Depth* as of December 23, 2011. Considering this information, it is my determination that *OSGi in Depth* was made available to the public at the Massachusetts Institute of Technology library shortly after its initial acquisition as of December 23, 2011.

57. Actual dissemination of a publication (such as *OSGi in Depth*) to interested persons exercising reasonable diligence is reflected by other publications, especially those of other authors, that make reference to it.

58. Attached hereto as **Appendix ALVES04** is (i) a screen capture, from the ACM Digital Library website, of bibliographic data for a research article entitled "A general extension system for event processing languages", published in *DEBS 11: Proceedings of the 5th ACM International Conference on Distributed Event-Based System,* 2011, as well as (ii) a true and complete copy of the research article of the same name that I retrieved from the same website. I secured these screen captures myself from https://dl.acm.org/doi/abs/10.1145/2002259.2002261. These are the type of records that experts in my field would reasonably rely upon when forming their opinion.

59. **Appendix ALVES04** indicates this research article was published on July 11, 2011. On PDF pages 2 and 13 of **Appendix ALVES04**, I note that this research article references: "[25] Alves, A., 2011, OSGi in Depth, Manning".

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0028

60. Attached hereto as **Appendix ALVES05** is (i) a screen capture, from the VDE Verlag website, of bibliographic data for a conference paper entitled "Smart Grid - Integration von Haushalten mittels Vernetzung und Energie-Community", authored by U. Trick, M. Steinheimer, P. Ruhrig, R. Tonjes and D. Holker, published in *VDE-Kongress 2012 – Intelligente Energieversorgung der Zukunft*, as well as (ii) a true and complete copy of a paper of the same name that I retrieved from the same website. I secured these screen captures myself from https://dl.acm.org/doi/abs/10.1145/2002259.2002261. These are the type of records that experts in my field would reasonably rely upon when forming their opinion.

61. **Appendix ALVES05** indicates this conference paper was presented at a conference in November 2012. On PDF page 7, I note that this conference paper references: "[22] Alves, A.: OSGi in Depth. Manning, 2011". I also note that the named authors of this conference paper entitled "Smart Grid - Integration von Haushalten mittels Vernetzung und Energie-Community" (**Appendix ALVES05**) are distinct from the named author of the *OSGi in Depth* book.

62. Attached hereto as **Appendix ALVES06** is (i) a screen capture, from the HAL Open Science website, of bibliographic data for a PhD thesis entitled "Embedding security policies into pervasive computing systems", authored by Pengfei Liu, and indicated as being submitted on March 12, 2013, as well as (ii) a

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0029

true and correct copy of selected pages of a PhD thesis that I retrieved from the same website. I secured these screen captures myself from https://theses.hal.science/tel-00799284/. These are the type of records that experts in my field would reasonably rely upon when forming their opinion.

63. **Appendix ALVES06** indicates this PhD thesis was submitted on March 12, 2013. On PDF page 10, I note that this conference paper references: "[34] A. de Castro Alves. *OSGi in Depth*. Manning Publications, 2011". I also note that the named author of this PhD thesis entitled "Embedding security policies into pervasive computing systems" (**Appendix ALVES06**) is distinct from the named author of the *OSGi in Depth* book.

3. Summary of My Opinions on *OSGi in Depth* (**Exhibit 002**)

64. My analysis of the copy of *OSGi in Depth* from the Library of Congress (**Exhibit 002**) informs my opinion that the content of **Exhibit 002** is authentic and is from a published book that is publicly accessible. The publisher's webpage (**Appendix ALVES07**), and a February 7, 2012 archived version of the publisher's webpage (**Appendix ALVES08**), both indicate that *OSGi in Depth* was published in December 2011, and showed that *OSGi in Depth* was available for purchase by interested persons from the publisher at least as early as that date. The MARC records in **Appendices ALVES02 and ALVES03** respectively indicate that the Rochester Institute of Technology library and the Massachusetts Institute

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0030

of Technology library each acquired *OSGi in Depth* as of December 23, 2011. The MARC record in **Appendix ALVES01** indicates that *OSGi in Depth* was made discoverable in the online catalog of the Library of Congress shortly after its initial acquisition as of February 24, 2012. Exemplary citation records show that *OSGi in Depth* was actually disseminated to interested persons exercising reasonable diligence (**Appendices ALVES05** and **ALVES06**) at least as early as November 2012 and March 2013.

65. Based on my review and analysis of these documents and records, it is my opinion that *OSGi in Depth* was cataloged and became publicly accessible at least by March 2012 from the Rochester Institute of Technology library and the Massachusetts Institute of Technology library, and at least by May 2012 from the Library of Congress library, (*see, e.g.*, ¶¶10, 64 (above)), and that *OSGi in Depth* was actually disseminated to interested persons exercising reasonable diligence at least by November 2012.

C. Authenticity and Public Availability of *OSGi in Action: Creating Modular Applications in Java* (Exhibit 003)

1. Authentication

66. **Exhibit 003** of my Declaration is a true and correct copy of select pages of *OSGi in Action: Creating Modular Applications in Java* by Richard S. Hall, et al. as held by the Library of Congress. These scans were secured from the library's collection. Researching this work, I searched the Worldcat.org website

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0031

for records, and a record at https://search.worldcat.org/title/751987617 identified the Library of Congress as one of the holding libraries. After verifying the holdings information in the online catalog of the Library of Congress (https://lccn.loc.gov/2011283913), I requested and obtained a scan copy of pages of *OSGi in Action: Creating Modular Applications in Java* from the Library of Congress through counsel. This library copy is presented as **Exhibit 003** in this declaration.

67. I have been informed by counsel and understand that Exhibit 1009 submitted by Petitioner in the IPR proceedings is the same true and correct copy of the select pages that were scanned from the copy secured from the Library of Congress's collection and that I attach as **Exhibit 003** to my Declaration.

68. PDF Page 1 shows the front cover, and page 32 shows the back cover of this book. The front cover shows the title, authors, and publisher to which this work belongs. It also shows a call number label of "QA 76 .73 .J38 -827 2011 Copy 1." The back cover shows a Library of Congress Cataloging-in-Publication (CIP) label that shows "0 028 813 668 3" as a Library of Congress control number of this record, shows the book's 13-digit ISBN (ISBN-13) is 978-1-933988-91-7 and the book's 10-digit ISBN (ISBN-10) is 1-933988-91-6, and indicates the companion website to this book (and a "free eBook for owners of this book") is at manning.com/OSGiinAction. PDF Page 2 is the title page that shows the same

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0032

title, authors and publisher information as the front cover. PDF Page 3 is the copyright page that shows a "2011" copyright date, and shows a Library of Congress barcode label showing "2011283913" as a Library of Congress control number of this record. This page also includes the full publisher information, and shows the 13-digit ISBN (that is also on the back cover). PDF Pages 4 to 37 are internal pages xx, 4-5, 179, 283, 331-334, 438-448, 457-464, and 471-476.

69. My research on *OSGi in Action: Creating Modular Applications in Java* discovered the publisher's webpage for this book at https://www.manning.com/books/osgi-in-action from the Manning Publications website. A screen capture of the publisher webpage is presented as **Appendix HALL07** in this declaration. Pages 1-2 of this screen capture shows *OSGi in Action: Creating Modular Applications in Java* was first published in "April 2011", shows the 13-digit ISBN (9781933988917) of the book (that is also on the back cover and copyright page of **Exhibit 003**), and also states that users "can see this entire book for free."

70. My research on *OSGi in Action: Creating Modular Applications in Java* also discovered an archived version of the publisher's webpage for this book in the Internet Archive records https://web.archive.org/web/20110701134343/http://www.manning.com/hall/. These are another type of record that experts in my field would reasonably rely

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0033

upon when forming their opinions. A screen capture of the archived version of the publisher webpage obtained from this Internet Archive link is presented as **Appendix HALL08** in this declaration. Page 1 of this screen capture shows *OSGi in Action: Creating Modular Applications in Java* was first published in "April 2011", shows the 10-digit ISBN (1933988916) of the book (which matches the back cover of **Exhibit 003**), contains a link to purchase the printed book (and the eBook) for $49.99, and shows users can freely access certain portions of *OSGi in Depth*, including Chapter 1, which the Internet Archive contains a record of this link to Chapter 1 at https://web.archive.org/web/20120127125720/http://www.manning.com/hall/OSGIiAsamplech1.pdf. PDF Pages 9-29 of **Appendix HALL08** include the PDF that is archived at this Chapter 1 link. I have closely compared pages 4-5 from the copy of Chapter 1 from the archived version of the publisher's website (**Appendix HALL08**) with those pages in the Library of Congress copy (**Exhibit 003**), and found that these pages in the library copy are the same as their corresponding pages in the publisher copy (at the archived version of the publisher's website).

71. Internet Archive is a digital library that archives scholarly materials, cultural artifacts, and other web resources on the Internet and provides the public with free access to its collections. I have used the Internet Archive in the past and am familiar with their practice of assigning an extended URL to an archived

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0034

resource in the format of http://www.archive.org/web/[Year in yyyy][Month in mm][Day in dd][Time code in hh:mm:ss]/[Archived URL] to show the point of time the archived file was captured. Because Internet Archive usually does not archive web resources the moment they appear on the Internet, and a web resource must be available first before it can be archived, the archive date of a file captured by the Internet Archive means that the archived file was available before, and definitely no later than, the archive date. **Appendix HALL08** shows the publisher webpage for *OSGi in Action: Creating Modular Applications in Java* was archived on "20110701134343" (*i.e.*, July 1, 2011, at 1 p.m., 43 minutes, 43 seconds). This archive date means the webpage would have been publicly discoverable on the Internet no later than July 1, 2011, and users would have been able to follow the links on the webpage to purchase a copy of *OSGi in Action: Creating Modular Applications in Java*.

72. **Exhibit 003** is in a condition that creates no suspicion about its authenticity. Specifically, the text on each consecutive page appears to flow seamlessly, and there are no visible alterations to the document. In addition, **Exhibit 003** is in the custody of a national library. I t is therefore my opinion that **Exhibit 003** is from a published book that is authentic and publicly available.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0035

2. Public Availability

73. Attached hereto as **Appendix HALL01** is a true and correct copy of the MARC record for *OSGi in Action: Creating Modular Applications in Java* as held by the Library of Congress. I secured this record myself from the library's public catalog. This is the type of record that experts in my field would reasonably rely upon when forming their opinions. The MARC record contained within **Appendix HALL01** includes fields denoting the title, author, publisher, copyright date ISBN numbers, and Library of Congress call number of *OSGi in Action: Creating Modular Applications in Java*, which match such information in **Exhibit 003**.

74. The 008 field of the MARC record in **Appendix HALL01** indicates the date of record creation. The 008 field of **Appendix HALL01** indicates the Library of Congress first acquired *OSGi in Action: Creating Modular Applications in Java* as of September 20, 2011. Considering this information, it is my determination that *OSGi in Action: Creating Modular Applications in Java* was made discoverable in the online catalog of the Library of Congress shortly after its initial acquisition as of September 20, 2011.

75. Attached hereto as **Appendix HALL02** is a true and correct copy of the MARC record for *OSGi in Action: Creating Modular Applications in Java* as held by the Rochester Institute of Technology library. I secured this record myself

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0036

from the library's public catalog. The MARC record contained within **Appendix HALL02** includes fields denoting the title, author, publisher, copyright date and ISBN numbers of *OSGi in Action: Creating Modular Applications in Java*, which match such information in **Exhibit 003**.

76. The 008 field of the MARC record in **Appendix HALL02** indicates the date of record creation. The 008 field of **Appendix HALL02** indicates the Rochester Institute of Technology library first acquired *OSGi in Action: Creating Modular Applications in Java* as of August 26, 2011. Considering this information, it is my determination that *OSGi in Action: Creating Modular Applications in Java* was made available to the public at the Rochester Institute of Technology library shortly after its initial acquisition as of August 26, 2011.

77. Attached hereto as **Appendix HALL03** is a true and correct copy of the MARC record for *OSGi in Action: Creating Modular Applications in Java* as held by the University of Chicago library. I secured this record myself from the library's public catalog. The MARC record contained within **Appendix HALL03** includes fields denoting the title, author, publisher, copyright date and ISBN numbers of *OSGi in Action: Creating Modular Applications in Java*, which match such information in **Exhibit 003**.

78. The 008 field of the MARC record in **Appendix HALL03** indicates the date of record creation. The 008 field of **Appendix HALL03** indicates the

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0037

University of Chicago library first acquired *OSGi in Action: Creating Modular Applications in Java* as of September 20, 2011. Considering this information, it is my determination that *OSGi in Action: Creating Modular Applications in Java* was made available to the public at the University of Chicago library shortly after its initial acquisition as of September 20, 2011.

79. Actual dissemination of a publication (such as *OSGi in Action: Creating Modular Applications in Java*) to interested persons exercising reasonable diligence is reflected by other publications, especially those of other authors, that make reference to it.

80. Attached hereto as **Appendix HALL04** is (i) a screen capture, from the Semantic Scholar website, of bibliographic data for a PhD thesis entitled "A Web of Things Application Architecture: Integrating the Real-World into the Web", authored by D. Guinard, submitted in 2011, as well as (ii) a true and complete copy of selected pages of a PhD thesis of the same name that I retrieved from the same website. I secured these screen captures myself from https://www.semanticscholar.org/paper/A-Web-of-Things-Application-Architecture%3A-the-into-Guinard/7a49bb63ce630e352551ce005a8c93e5ad41c6cf?sort=relevance&queryString=Hall&citedSort=relevance&citedPage=11. These are the type of records that experts in my field would reasonably rely upon when forming their opinion.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0038

81. **Appendix HALL04** indicates this PhD thesis was submitted in 2011. On PDF pages 3 and 41 of **Appendix HALL04**, I note that this PhD thesis references: “Richard Hall, Karl Pauls, Stuart McCulloch, and David Savage. OSGi in Action: Creating Modular Applications in Java. Manning Publications, April 2011”. I also note that the named author of this PhD thesis entitled "A Web of Things Application Architecture: Integrating the Real-World into the Web" (**Appendix HALL04**) is distinct from the named authors of the *OSGi in Action: Creating Modular Applications in Java* book.

82. Attached hereto as **Appendix HALL05** is (i) a screen capture, from the IEEE Xplore website, of bibliographic data for a conference paper entitled "Establishing a Smart Grid Node Architecture and Demonstrator in an Office Environment Using the SOA Approach", authored by Dagmar Koß, Florian Sellmayr, Steffen Bauereiß, Denis Bytschkow, Pragya Kirti Gupta, Bernhard Schatz, and published in the *Proceedings of the 2012 First International Workshop on Software Engineering Challenges*, as well as (ii) a true and complete copy of a conference paper of the same name that I retrieved from the same website. I secured these screen captures myself from https://ieeexplore.ieee.org/abstract/document/6225710. These are the type of records that experts in my field would reasonably rely upon when forming their opinion.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0039

83. **Appendix HALL05** indicates this conference paper was presented at a conference in June 2012, and added to IEEE Xplore on June 28, 2012. On PDF page 11, I note that this conference paper references: "[22] R. S. Hall, K. Pauls, S. McCulloch, and D. Savage, *OSGi in Action: Creating Modular Applications in Java*. Manning, 2011". I also note that the named author of this conference paper entitled "Establishing a Smart Grid Node Architecture and Demonstrator in an Office Environment Using the SOA Approach" (**Appendix HALL05**) is distinct from the named authors of the *OSGi in Action: Creating Modular Applications in Java* book.

84. Attached hereto as **Appendix HALL06** is (i) a screen capture, from the International Journal of Computer Applications website, of bibliographic data for a research article entitled " Decentralized Principles: New Modular Software Development Principles, a Robust Object Oriented Approach", authored by GholamAli Nejad and HajAli Irani, and published in the *International Journal of Computer Applications (Volume: 44, Number 13, April 2012)*, as well as (ii) a true and correct copy of a research article of the same name. I secured these screen captures myself from https://www.ijcaonline.org/archives/volume44/number13/6324-8671/. These are the type of records that experts in my field would reasonably rely upon when forming their opinion.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0040

85. **Appendix HALL06** indicates this paper was published by the Foundation of Computer Science in April 2012. On PDF page 10, I note that this paper references: "[23] R. S. Hall, K. Pauls, S. McCulloch, D. Savage, OSGi in Action, Manning Publications, 2011". I also note that the named author of this research article entitled " Decentralized Principles: New Modular Software Development Principles, a Robust Object Oriented Approach" (**Appendix HALL06**) is distinct from the named authors of the *OSGi in Action: Creating Modular Applications in Java* book.

3. Summary of my Opinions on *OSGi in Action: Creating Modular Applications in Java* (**Exhibit 003**)

86. My analysis of the copy of *OSGi in Action: Creating Modular Applications in Java* from the Library of Congress (**Exhibit 003**) informs my opinion that the content of **Exhibit 003** is authentic and is from a published book that is publicly accessible. The publisher's webpage (**Appendix HALL07**), and a February 7, 2012 archived version of the publisher's webpage (**Appendix HALLX08**), both indicate that *OSGi in Action: Creating Modular Applications in Java* was published in April 2011, and showed that *OSGi in Action: Creating Modular Applications in Java* was available for purchase by interested persons from the publisher at least as early as that date. The MARC records in **Appendices HALL02** and **HALL03** respectively indicate that the Rochester Institute of Technology library and the University of Chicago library acquired

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0041

OSGi in Action: Creating Modular Applications in Java as of October 26, 2011 and September 20, 2011 respectively. The MARC record in **Appendix ALVES01** indicates that *OSGi in Action: Creating Modular Applications in Java* was made discoverable in the online catalog of the Library of Congress shortly after its initial acquisition as of September 20, 2011. Exemplary citation records (**Appendices HALL04**, **HALL05**, **HALL06**) show that *OSGi in Action: Creating Modular Applications in Java* was actually disseminated to interested persons exercising reasonable diligence at least as early as December 2011, April 2012 and June 2012.

87. Based on my review and analysis of these documents and records, it is my opinion that *OSGi in Action: Creating Modular Applications in Java* was cataloged and became publicly accessible by no later than December 2011 from the Library of Congress and the Rochester Institute of Technology library, and at least by January 2012 from the University of Chicago library, (*see, e.g.*, ¶¶10, 64 (above)), and that *OSGi in Action: Creating Modular Applications in Java* was actually disseminated to interested persons exercising reasonable diligence at least by December 2011.

III. CONCLUSION

88. My findings and opinions set forth in this Declaration are based on my work and examination to date. I may continue my examinations in view of additional documents or other factual evidence over the course of this proceeding

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0042

that may necessitate supplementing and/or refining my opinions. I reserve the right to add to, alter, or delete my opinions and my Declaration upon discovery of additional information. I reserve the right to make such changes as necessary.

89. In signing this Declaration, I understand that the Declaration will be filed as evidence in a contested case before the Patent Trial and Appeal Board of the United States Patent and Trademark Office. I acknowledge that I may be subject to cross-examination in this case and that cross-examination will take place within the United States. If cross-examination is required of me, I will appear for cross-examination within the United States during the time allotted for cross-examination.

90. I declare under penalty of perjury that the foregoing is true and correct. I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0043

Dated: January 27, 2025



June Ann Munford

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0044

Exhibit 001

Dependability as a Cloud Service - A Modular Approach

Jan S. Rellermeyer[1]
[1]*IBM Research,*
Austin Research Lab,
Austin, TX 78758
rellermeyer@us.ibm.com

Saurabh Bagchi[1,2]
[2]*Purdue University,*
School of Electrical and Computer Engineering,
West Lafayette, IN 47907
sbagchi@purdue.edu, sbagchi@us.ibm.com

***Abstract*—Failures of services on cloud platforms are only to be expected. To deal with such failures, one is naturally inclined to use the traditional measure of replication. However, replication of services on distributed cloud platforms poses several challenges that are not well met by today's Java middleware systems. These challenges are the need to isolate state in the application components so that easy migration and recovery are possible and the requirement for client transparency when dealing with different replicated service instances. For example, Java Enterprise Edition (JEE) makes it difficult to have transparent replication of services due to the above two reasons plus the fine-grained nature of interactions between its components (the Enterprise Java Beans). In this paper, we show parts of the design of OSGi, a specification defining a dynamic component system in Java, that make it suitable for the above task. We then propose two extensions to OSGi which will allow exposing and exporting application component state and transparent invocation of service instances. These two together can enable easy replication and recovery from failures in cloud environments. We show through experiments that our prototype can migrate a failed service quickly enough to a new machine so that a client experiences only a moderate increase in service invocation time during system recovery.**

I. Introduction

A decision to move away from privately owned computer resources and towards cloud computing is affected by a variety of factors, dependability being one of them. Dependability is usually defined through its attributes of availability, reliability, safety, integrity, and maintainability [1]. In a complex system, the overall dependability of the system depends on the corresponding attributes of its components. What changes in cloud computing is that one important component—the platform—is no longer under one's own control. Depending on the type of the cloud service, the platform may mean one of several things; at least it includes the physical machine but may also include higher layers like a complete runtime system. This arguably outsources part of the maintenance problem while introducing new challenges. Most importantly, the underlying physical resources (i.e., the machines and the network) are shared between multiple customers and applications and cannot be actively managed. This is often perceived to be a threat to the overall dependability [2], [3], [4].

Typical ways of mitigating these threats in a cloud computing environment revolve around introducing redundancy into the system, especially for increased availability and reliability. Since adverse incidents can always render a single instance of a functional unit unusable, the hope is that by creating multiple copies of the component a sufficient number of copies always survive to deliver the service without degradation. This idea is neither new nor specific to cloud computing and has been successfully applied many many times in earlier distributed systems such as clusters or grids [5], [6]. In cloud computing, however, the effectiveness of redundancy benefits from two particular factors. First, the elasticity of the resource fabric makes redundancy more agile and more affordable. This means that it is possible to acquire additional resources for increasing the degree of redundancy in response to perceived threats much more easily than it can be done in traditional hosted environments. Second, the resource fabric is typically globally distributed, thereby allowing intelligent placement to guard against high-impact outages that would be much harder to circumvent in a hosted environment. By spanning multiple geographic zones, a system can survive even an outage of an entire data center that could be caused by a natural disaster or a far-reaching power outage.

Unfortunately, not every component of a complex system can easily profit from redundancy. Components that have state in them cannot be easily split up for purposes of replicating the parts to different replication degrees and executed on different machines. There have been prior attempts to address this by structuring web services such that most of the components are stateless and the few stateful components expose their state in an orderly manner. Thereby, it can be handled through special, and more expensive, mechanisms such as database replication. However, we discern that this kind of structuring of applications is becoming harder to do. Web services are becoming contextualized by various factors including the user, the client environment, the results of prior requests, and interactions with coordinating and collaborating web services. For example, with the advent of social networks, the services are contextualized by the user profiles and for a rich interactive user experience, the

978-1-4673-2266-9/12/$31.00 ©2012 IEEE

Authorized licensed use limited to: Rochester Institute of Technology. Downloaded on September 27,2024 at 13:30:55 UTC from IEEE Xplore. Restrictions apply.

state (in this case, the profile) is driving the majority of the functionality. *Therefore, our position is that existing mechanisms for structuring applications for purposes of reliability will no longer work with emerging web services, that are being hosted on the cloud.*

The effectiveness of elasticity depends to a high degree on the coupling of components. Systems with fine granular components tend to suffer from a high degree of entanglement, hence limiting the flexibility of selectively replicating components since all tightly-coupled components usually have to reside on the same host. Consider for example, Enterprise Java Beans running on a JEE middleware are typically structured to have final granularity and with frequent invocations between them, both of which highlight the challenges of replication for cloud environments.

In this paper, we discuss how a dependability service can be built on the OSGi framework. OSGi is a set of specifications [7] that defines a dynamic component system for Java. These specifications enable a development model where applications are *dynamically* composed of many different reusable components. The OSGi specifications enable components to hide their implementations from other components while communicating through services, which are objects that are specifically shared between components. We show why a module system for Java, like OSGi, makes our goal simpler to achieve and then show how the dependability service can be architected using features already provided by OSGi. We propose an extension to the OSGi standard that acts as a dependability service, which we call CLOUDDEP. The service provides the following dependability properties:

1) Re-deployment: Migrating the application to a new physical machine or a different data center when the underlying platform fails.
2) Clustering: Eliminating single points of failure in the system by replicating functional units.
3) Elasticity and Load-balancing: Distributing the load to a flexible pool of resources to avoid overloading single instances.

II. SOFTWARE MODULARITY AND DEPENDABILITY

Software modules are self-contained functional units of encapsulation. They contain data and functions, objects, or other entities of the programming languages used to implement them. Ideally, the correctness of a modular system depends only on the correctness of its modules and the correctness of a single module does not depend on other modules [8]. The requirement of being self-contained, however does not automatically imply full isolation. Many implementations of module technology allow modules to be *declaratively self-contained*, i.e., they not only rely on their own content but can additionally declare dependencies to other modules and have access to their accessible content. In such setups, the effects of errors occurring in a single module are still restricted to cause faults in this module or any module that has a declared dependency but not in unrelated modules.

This property has been successfully exploited to increase the fault-tolerance of systems. For instance, in the Actor model [9] the modular units are fully isolated and can only communicate through explicit message passing.

Micro Kernel Operating Systems [10] restrict themselves to implement only the minimally necessary functionality in the kernel and outsource the remaining responsibilities into separate user-space modules. Thereby, faults in one module cannot affect other modules or the privileged kernel.

Microreboots [11] describe a methodology for recovering a system by rebooting only a subset of affected components, in the concrete example Enterprise Java Beans (EJBs). The design approach is to keep application state carefully parceled away in separate state stores which are only accessed through well-defined interfaces. Thus, the application components can be recovered by rebooting them (since the rebooting is fast in the absence of state, it is called microrebooting), while the state store can persist across reboots. This approach shares some similarities with our proposed approach. However, in order to be micro-rebootable, the components need to be well-isolated and stateless except for externalized application state kept in specialized state stores. Such compartmentalization is non-trivial to do in the context of JEE because of the reasons mentioned earlier (tight coupling and fine granular decomposition).

III. OSGI

A. OSGi Background

OSGi is a set of standards for modularity on the Java virtual machine released and maintained by the OSGi Alliance. Originally, it was developed by several major vendors to solve an availability problem on embedded home gateway machines [12]. The intention was that multiple software components should be able to co-exist on the same hardware while each of the software packages needs to run continuously and over long periods of time, even in scenarios that require periodic maintenance updates to the software. To enable dynamic updates without stoppage, it aimed at overcoming the traditional entanglement of small-granular objects and formed larger modules that can be managed individually and composed to form applications. OSGi defines a unit of modularity as the *Bundle*, which is a deployable Java JAR file enriched with additional meta-data. Most importantly, this meta-data describes which packages are exported, i.e., accessible to other bundles, and which packages it needs to import from other bundles. Recently, OSGi has successfully entered the domain of enterprise software and most major Java application servers (e.g., IBM WebSphere, JBoss, Oracle WebLogic, Spring Dynamic Module Platform) are by now internally based on the OSGi technology.

Authorized licensed use limited to: Rochester Institute of Technology. Downloaded on September 27,2024 at 13:30:55 UTC from IEEE Xplore. Restrictions apply.

The OSGi runtime environment, the *Framework*, provides the user with mechanisms for installing and controlling the life-cycle of bundles at runtime. Applications are typically composed out of many different bundles. These bundles can interact in a tightly-coupled way by depending on packages provided by other bundles. *Tight coupling* in this context means that the dependencies are required to resolve the bundle and failure to provide the dependencies renders the bundle inoperable. Another way of interaction is through loosely-coupled services. *Loose coupling* means that service consumers only rely on the service interfaces under which the service has been registered with the OSGi framework but not on any knowledge of implementation details. Bundles can consequently query the framework for available services and retrieve the service instance, which is an ordinary Java object in OSGi. The result of a query is either a reference, a set of references from which the client can pick the best matching service, or the information that no such service is available. Since the consumers only rely on the service interfaces but not on any concrete implementation, bundles can potentially operate even in the absence of certain services. OSGi applications are expected to actively deal with dynamism in the setup. For instance, a service can certainly fulfill its purpose in the absence of a log service but if it has logging support, e.g., for debugging, it should enable it whenever a log service becomes available and disable it when the service becomes unavailable.

The modularity in OSGi solves a major issue with creating elastic applications for the cloud. Cloud resources are inherently dynamic in their nature and undergo frequent changes either due to explicit management operations (adding and removing resources) or due to their volatility (sharing effects, failures). Therefore, we believe that monolithic software incapable of dynamic adaptation cannot effectively run in such an environment. In fact, the compositional approach of modular software—traditionally applied to a single runtime system—is key to building scalable and dependable systems across a varying set of machines in the cloud [13], [14]. OSGi particularly facilitates composition and localized behavior by separating tightly coupled and loosely-coupled interaction between modules into two different concepts: package dependencies and services, respectively. Thereby, when services are used as a design element, the resulting systems are less entangled and more flexible.

OSGi was originally designed for managing software on a single Java virtual machine. In prior work, we have shown that the OSGi model can be used as an application model for building distributed systems by turning the services into potential boundaries of distribution [15] and adding simple support for creating redundancy by clustering stateless services on multiple machines [16]. However, little attention has so far been paid to the problem of managing stateful services in distributed deployments like clusters or clouds.

B. How OSGi Configuration Admin Enables Restarts

A traditional OSGi framework (running on a single JVM) already supports restarts of the runtime systems as well as a form of partial restart for bundle updates. This means that whenever a bundle has been updated and the operator of the framework requests a refresh of the system, only those bundles immediately affected by the update (this is the bundle itself and all bundles relying on the updated bundle through tightly-coupled package dependencies) are restarted whereas the remainder of the system is unaffected. However, only the compositional state of the application is preserved across restarts. This is the information about which bundles have been installed when the framework was last running and in which life-cycle state each individual bundle has been in (installed, resolved, starting, active, stopping, uninstalled). Every bundle then gets started again but is itself responsible for restoring any internal state, which includes services that it has bound to.

The only platform support in traditional OSGi for externalizing bundle state is the Configuration Admin service, an optional service described in the OSGi Compendium Specifications [17]. The configuration admin is a service in the system capable of managing and persisting configuration data in a central location. Services that want either to be externally configurable or want to persist their configuration data across restarts can register as a *Managed Service*. This expresses its intent to receive configuration updates and gives the system an interface through which it can push updates to the service. Every managed service is expected to identify itself though a persistent identifier (PID) and announce this PID with its initial registration. The config admin uses the PID of a service as a primary key for associating configuration data to services. Configuration events and changes are signaled to the managed service asynchronously, regardless of whether they are used by the service itself (as illustrated in Figure 1) or an external bundle.

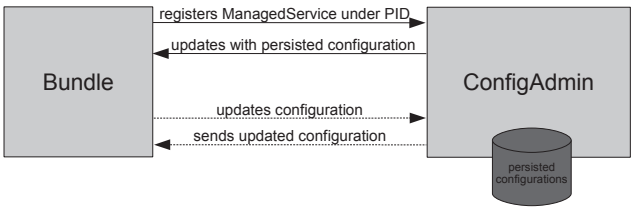


Figure 1. Interaction between Bundle and Configuration Admin

In the context of cloud computing, the most interesting and relevant property of the config admin is the concept of a service identity. When migrating a service to a new machine, it has to be assured that the newly created service instance can fully take over the role of the previous instance. Therefore, stateful services implicitly build up their own identity based on their internal state and their

Authorized licensed use limited to: Rochester Institute of Technology. Downloaded on September 27,2024 at 13:30:55 UTC from IEEE Xplore. Restrictions apply.

history. Migration and failover can be considered a "micro-reboot" of the service on a different machine. If all state could be stored as configuration data then a distributed version of the configuration admin would turn managed services into migratable services just as the ConfigAdmin allows stateful services to become restartable. In practice, the storage capacity of the configuration admin and its unclear consistency model are prohibitive to this approach. For instance, a configuration update always fully overwrites any previous configuration so that when two writers are updating disjoint parts of the configuration concurrently the last writer wins and the update from the earlier writer is lost.

However, just as systems like Zookeeper [18] are used in cloud deployments to manage configuration, a cloud-scale version of an OSGi ConfigAdmin can be used to dynamically bind service instances to a richer instance of storage to externalize the service state, e.g., a database system or a key-value store. The challenge of building such a distributed ConfigAdmin for cloud deployments is to coordinate it with the distribution of services. It is not feasible to have a single scope for configurations across all nodes since this would prohibit multiple redundant copies of the same service. When a service needs to be migrated to a different machine, however, the configuration has to be migrated with the service.

C. How OSGi Remote Service Admin Enables Non-local Execution

The Remote Service Admin aims at integrating distribution into the OSGi platform and has been part of the OSGi Enterprise Specifications [19]. This service encapsulates the mechanism for importing remote services into the scope of a local OSGi framework and exporting local OSGi services through distribution providers. When a remote service is imported, a local proxy for the remote service is created and associated with the distribution provider. Some distribution providers like our R-OSGi [15] system are able to operate transparently, i.e., neither client nor service have to be rewritten to operate in a remote setup.

The API of the remote service admin is entirely imperative. Given a known URI or other identifier, the admin provisions the local proxy and handles binding to the remote service reachable through this address, as long as the service is reachable. Finding a service in the network and re-binding in the event of a failure is handled by different components. All policy decisions are encapsulated in a Topology Manager which, e.g., is able to detect clients that are querying the local framework for services and make use of service discovery protocols to find matching services. However, the specifications do not mandate any particular functionality of the topology manager but only specify the interfaces through which a possible topology manager can interact.

For a cloud setup with dependability requirements, the presence of an appropriate topology manager is paramount.

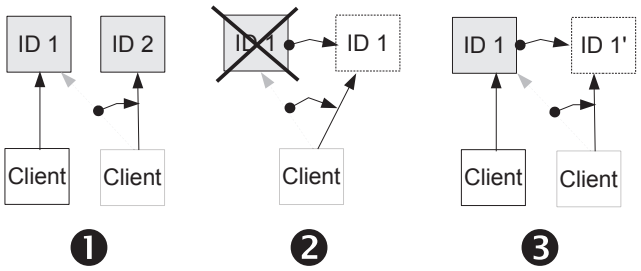


Figure 2. Different Use Cases for a Dependability Service. 1: Load balancing, 2: Failover, 3: Elastic Scalability.

In the following section, we describe our service CLOUDDEP that implements policies to ensure availability of services by acting as both a topology manager and a distributed configuration admin.

IV. TOWARDS A DEPENDABILITY SERVICE

The new service (that we call CLOUDDEP) is designed to turn OSGi services into elastic units of deployment which can be migrated and replicated across multiple cloud nodes to increase the availability and reliability of the administered service. All that is required from the administered service is that it persists all of its internal state either directly or indirectly through configurations and listens to configuration updates by implementing the Managed Service interface of the configuration admin specification. CLOUDDEP acts as a global entity in the system (which can be implemented by a centralized backend or a distributed architecture) and takes over the role of a local configuration admin and a local topology manager on each node of the cloud deployment. It also monitors the service invocations to detect failures. We illustrate the features of CLOUDDEP through three usage cases.

A. Load Balancing

We consider a simple setup with a set of identical (possibly stateful) remote service and a set of service clients as illustrated by case ❶ in Figure 2. Thanks to its facet of being a topology manager, CLOUDDEP has knowledge of the remote service relationship and was involved in the setup of the binding. It had detected the local service requests of the client bundles (e.g., through service lookup hooks as described in the remote service admin specification) and has instructed the distribution provider to create the binding to one of the known remote service instances. Furthermore, it has, either due to cooperation of the distribution provider or by wrapping its created service proxy, instrumented the local proxy to gather information about the frequencies of service invocation and response times. In the event that one remote service instance becomes overloaded, detectable by an increase of invocations and an increase of the response time, CLOUDDEP can re-bind a client to a different service

Authorized licensed use limited to: Rochester Institute of Technology. Downloaded on September 27,2024 at 13:30:55 UTC from IEEE Xplore. Restrictions apply.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0049

instance, thereby re-distributing the load among the service instances.

B. Failover

We now consider a setup with one or more clients and a single remote service. In the event of a service failure (e.g., an uncaught runtime exception or the node hosting the remote service becoming unavailable) CLOUDDEP has captured the identity of the faulty service through its configuration admin facet. It can thereby instantiate a new service by starting the corresponding bundle on a new node and inject the identity from the failed service into the new instance, as shown in case ❷. This process is akin to what a microreboot would do on a single node but here applied to a distributed setup. By applying the same rebinding as done for load balancing, the existing client will be forced to now interact with the new service instance. If the latency of service migration is an issue, CLOUDDEP can be configured to keep hot-spare instances of the service. These are running but unconfigured instances that have not yet received an identity. In the event of a service failure, CLOUDDEP can directly inject the identity into a hot-spare instance and create a more seamless failover.

C. Elastic Scalability

The third example combines the load balancing and the failover scenarios to enable elastic scalability of a remote service. In the event of a service over-utilization, CLOUDDEP can start a new instance of the service and bind some of the clients to the new service (see ❸ in Figure 2). A new identity is forked by CLOUDDEP from an existing configuration and can then develop independently from the original sibling. Scaling back to a smaller number of service instances can be achieved by applying the failover strategy: moving all clients of a service to a different existing instance and then dropping the service instance.

V. PROTOTYPE AND EXPERIMENTAL EVALUATION

We have implemented a prototype of CLOUDDEP that replicates configuration and manages failover for OSGi services in a cloud setup. The prototype uses R-OSGi as a distribution provider and cooperates with the protocol and the proxy handler to detect failures or service outages. The detection of failure is done through either a simple timeout or an exception caused by the termination of a connection. Since we induce failures in the experiments through terminating the connection, the latter detection is triggered. CLOUDDEP can furthermore either proactively start idle and unconfigured redundant instances of the service on a backup machine or reactively start a new instance of the service on demand. In either case, the identity of the failed service is injected into the new instance and the existing binding between the client and the no longer available service is transparently altered to point to the new instance. The prototype blocks the failed service request and replays the request to the new service instance as soon as it becomes available. We therefore assume a fail-stop model in our setup.

	Microreboot	Migration
avg. inv. time (T)	$7.23\mu s \pm 1.06\mu s$	$1.22ms \pm 0.15ms$
avg. T during failover	$266.29\mu s \pm 34.90\mu s$	$4.01ms \pm 0.66ms$
relative overhead	3581%	229%

Table I
SERVICE INVOCATION LATENCIES OF THE TREND SERVICE

For the experiment, we use a trend service that updates itself by periodically polling the most frequently used #hashtags from Twitter. The service is clustered by geographic regions, each instance of the service is configured to handle a specific region that is set through its configuration. Since the service only handles the ten most relevant results, it can afford to use the Configuration Admin interface to persist its internal state and does not rely on, e.g., an external blob storage. In a realistic setup, the service would process the results and, for instance, enrich the data with context links retrieved from Google Search or Wikipedia.

We use a single instance of the service and a single client that polls the service every second to receive the latest trends. After one minute, we take the service down and let CLOUDDEP handle the failure based on a strategy that we have pre-configured.

In the first experiment, we have both the client and the service running in the same Java Virtual Machine and on the same OSGi framework. Our failure recovery strategy resembles a microreboot of the service, i.e., we stop the running service and restart it. In the second experiment, we run client and service in different cloud instances and on different Java Virtual Machines and additionally keep a hot-spare VM for the failover. When the original service fails, we instantiate a new service instance on the hot-spare that becomes enabled when CLOUDDEP injects the configuration (i.e., geographic location and previously determined trends) into the instance.

Table I summarizes the measured average service invocation time during normal operation and during the failover. In the case of the microreboot, the invocation time during failover increases by a factor of 35 during failover. The reason for this high increase is that in OSGi clients get a direct reference to the service object and thereby a regular service invocation does not differ from any other method call in terms of latency. Enterprise Java Beans, in contrast, are called through reflection. Furthermore, a bundle restart is potentially more expensive than a microreboot of a single Enterprise Bean. However, the overall latency during failover is still in the range of microseconds, which is very likely acceptable for the large majority of applications.

In the distributed case the baseline of an average regular

Authorized licensed use limited to: Rochester Institute of Technology. Downloaded on September 27,2024 at 13:30:55 UTC from IEEE Xplore. Restrictions apply.

service invocation is much higher and is in our setup in the order of milliseconds. However, the failover increases the invocation time by only a factor of three. In practice, this factor will vary with the amount of configuration data that needs to be migrated to restore the service identity and the network used but more complex services are likely to also have higher baseline invocation times. In both the local and the distributed case, only a single service invocation is temporarily blocked and experiences a higher latency. Further invocations continue to operate at normal latency.

The results illustrate the viability of our approach for a large class of applications. Microreboots are an option for recovery from failures, especially when the service is already accessed from a different host. In this case, the latency of the microreboot is within the noise of a remote service invocation. When entire nodes fail or become unreachable, migration of services to different machines are also feasible and do not affect the client significantly. We have, however, assumed that the cloud instance for failover is already instantiated. In our setup, we used an Eucalyptus cluster that requires in average 85 seconds to boot up a new instance to the point where we can interact with it. This latency is unlikely to be acceptable for cloud services but is a common problem of IaaS solutions.

VI. Conclusion

We expect dependability to become an increasing problem for cloud services. Our premise is that traditional ways of structuring such systems provide insufficient flexibility and cannot deal with failures or elasticity requirements as they occur in cloud setups. We have therefore proposed a modular, loosely-coupled approach for building cloud services and have illustrated the concrete challenges using the example of OSGi, a widely used standard for modularity for the Java language. The result of this effort is CloudDep, a dependability service for OSGi that takes care of countering service failures by migrating the identity of the service to a new machine. The experimental results of our prototype implementation show that migration to an existing node is feasible and the impact on the service invocation time is acceptable for most applications.

References

[1] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, 2004.

[2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, Apr. 2010.

[3] "Cloud Computing Incidents Database." [Online]. Available: http://wiki.cloudcommunity.org/wiki/CloudComputing:Incidents_Database

[4] K. Joshi, G. Bunker, F. Jahanian, A. van Moorsel, and J. Weinman, "Dependability in the cloud: Challenges and opportunities," in *Dependable Systems Networks, 2009. DSN '09. IEEE/IFIP International Conference on*, 29 2009-july 2 2009, pp. 103 –104.

[5] H. Stockinger, A. Samar, K. Holtman, B. Allcock, I. Foster, and B. Tierney, "File and object replication in data grids," *Cluster Computing*, vol. 5, no. 3, pp. 305–314, 2002.

[6] J. Abawajy, "Fault-tolerant scheduling policy for grid computing systems," in *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*. IEEE, 2004, pp. 238–244.

[7] OSGi Alliance, *OSGi Service Platform, Core Specification Release 4, Version 4.2*, 2009.

[8] B. Meyer, *Object-Oriented Software Construction*, 1st ed. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1988.

[9] C. Hewitt, P. Bishop, and R. Steiger, "A universal modular ACTOR formalism for artificial intelligence," in *Proceedings of the 3rd international joint conference on Artificial intelligence*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1973, pp. 235–245.

[10] J. Liedtke, "On micro-kernel construction," in *Proceedings of the fifteenth ACM symposium on Operating systems principles*, ser. SOSP '95. New York, NY, USA: ACM, 1995, pp. 237–250.

[11] G. Candea, S. Kawamoto, Y. Fujiki, G. Friedman, and A. Fox, "Microreboot - a technique for cheap recovery," in *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, ser. OSDI'04, 2004.

[12] P. Kriens, "How osgi changed my life," *ACM Queue*, vol. 6, no. 1, Jan. 2008.

[13] C. Matthews and Y. Coady, "Virtualized recomposition: Cloudy or clear?" *Software Engineering Challenges of Cloud Computing, ICSE Workshop on*, vol. 0, pp. 38–43, 2009.

[14] J. S. Rellermeyer, M. Duller, and G. Alonso, "Engineering the cloud from software modules," in *ICSE-Cloud 09: Proceedings of the Workshop on Software Engineering Challenges in Cloud Computing (in conjunction with ICSE 2009)*, Vancouver, Canada, 2009.

[15] J. S. Rellermeyer, G. Alonso, and T. Roscoe, "R-osgi: Distributed applications through software modularization," in *Middleware 07: 8th ACM/IFIP/USENIX International Middleware Conference*, Newport Beach, CA, USA, 2007.

[16] J. S. Rellermeyer, G. Alonso, and T. Roscoe, "Building, deploying, and monitoring distributed applications with eclipse and r-osgi," in *ETX 07: Fifth Eclipse Technology eXchange (ETX) Workshop (in conjunction with OOPSLA 2007)*, Montreal, Canada, 2007.

[17] OSGi Alliance, *OSGi Service Platform, Service Compendium Release 4, Version 4.2*, 2009.

[18] Apache Foundation, "Apache zookeeper," http://zookeeper.apache.org, 2008.

[19] OSGi Alliance, *OSGi Service Platform Enterprise Specifications, Release 4, Version 4.2*, 2010.

Authorized licensed use limited to: Rochester Institute of Technology. Downloaded on September 27,2024 at 13:30:55 UTC from IEEE Xplore. Restrictions apply.

Exhibit 002

QA 76
.73
.J38
A458
2012
Copy 1

OSGi IN DEPTH



Alexandre de Castro Alves

FOREWORD BY DAVID BOSSCHAERT

MANNING

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0053

OSGi in Depth

ALEXANDRE DE CASTRO ALVES



MANNING
SHELTER ISLAND

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0054

QA76
.73
.J38 A158
2012
Copy 1

For online information and ordering of this and other Manning books, please visit www.manning.com. The publisher offers discounts on this book when ordered in quantity. For more information, please contact

Special Sales Department
Manning Publications Co.
20 Baldwin Road
PO Box 261
Shelter Island, NY 11964
Email: orders@manning.com

©2012 by Manning Publications Co. All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by means electronic, mechanical, photocopying, or otherwise, without prior written permission of the publisher.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in the book, and Manning Publications was aware of a trademark claim, the designations have been printed in initial caps or all caps.

Recognizing the importance of preserving what has been written, it is Manning's policy to have the books we publish printed on acid-free paper, and we exert our best efforts to that end. Recognizing also our responsibility to conserve the resources of our planet, Manning books are printed on paper that is at least 15 percent recycled and processed without elemental chlorine.

Manning Publications Co.
20 Baldwin Road
PO Box 261
Shelter Island, NY 11964

Development editor: Sebastian Stirling
Copyeditor: Linda Recktenwald
Poofreader: Andy Carroll
Typesetter: Marija Tudor
Cover designer: Marija Tudor

ISBN: 9781935182177
Printed in the United States of America
1 2 3 4 5 6 7 8 9 10 – MAL – 17 16 15 14 13 12 11

Library of Congress
2011 284960

foreword

After many years in ClassPath-land, the Java industry at large is finally starting to see the need for modularity in software development and deployment. There are many reasons for this. Maybe you just want to get out of the *Jar Hell*, where you're not quite sure anymore where a particular class comes from in your deployment. Other use cases for modularity center around providing isolation in a multi-tenancy context.

But there are more reasons. A modular approach makes developing software more scalable as modules tend to be highly focused with a clear demarcation of responsibility. A module's internals are inaccessible to the outside, which means that a modular approach tends to lead to well-defined APIs that better allow for concurrent development. It enables structuring of development teams such that parallel development of modules, which together form the application, is achievable.

The OSGi specifications provide a very mature, stable, and comprehensive modularity solution. The OSGi Core specification defines the OSGi framework, addressing modularity, lifecycle, services, and security aspects. Together these enable a dynamic system where *bundles*, the OSGi name for modules, are often remarkably reusable, properly encapsulated, and loosely coupled. The OSGi service registry enables an elegant plug-in model where consumers don't need to be preconfigured with any expected service provider implementation. The OSGi service model allows for services to be changed dynamically at runtime without the need to modify their consumers.

In this age of cloud computing, a system needs to be dynamically adaptable, highly manageable, and easily maintainable. OSGi technology facilitates all this alongside what is generally an extremely light infrastructure footprint. Runtime metadata and

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0056

framework management is available through a standard API or via JMX, if enabled, and because bundles generally have clearly defined purposes, maintenance is easier and more localized—there's no more *big ball of mud* that needs to be dealt with.

OSGi is the only standards-based solution to modularity today, and given that it has been around since the late 1990s, it's a well-matured and very stable technology. A number of highly popular open source projects provide OSGi framework implementations today, and a number of commercial implementations are also available. OSGi is being used in contexts from embedded, residential, and mobile devices to highly scalable and performant server systems. Additionally, it is the infrastructure behind many rich client applications, of which the Eclipse IDE is probably the most well-known.

In this book, Alex Alves looks in detail at many of the OSGi Core concepts while also elaborating on a number of vital technologies from the OSGi Compendium and Enterprise specifications. As cochair of the Enterprise Expert Group, I'm particularly pleased to see a number of Enterprise OSGi technologies covered. You'll find chapters about OSGi remote services (note a very interesting cloud computing section in this chapter), JDBC and JPA, transactions, JNDI integration, and JMX support, while the Configuration Admin and Event Admin services are also covered. Last but not least, you'll find coverage of OSGi Blueprint, a specification inspired by the Spring Framework aimed at using and creating OSGi services in a simple and user-friendly way.

This is a book that both covers the high-level big-picture architecture topics as well as the details involved in getting things working on a practical level. It will give you a deep understanding of OSGi and will provide you with the knowledge you need to utilize OSGi to the full.

DAVID BOSSCHAERT
PRINCIPAL SOFTWARE ENGINEER, JBOSS BY RED HAT
OSGi ENTERPRISE EXPERT GROUP COCHAIR

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0057

about this book

In this book, I show how the OSGi technology can be used to write better software, and in particular, how OSGi can be used to write better platforms for the development of better software.

Most importantly, I focus on real problems and on how OSGi can be used to solve them. Instead of just explaining OSGi's API for modularization, I first show you the problems that arise due to the lack of modularization. Instead of simply giving you OSGi's transaction API, I show you why and when you need to use transactions, and what the implications are of using transactions in a modularized service-oriented fashion in OSGi. As you read this book, you'll acquire in-depth knowledge of OSGi, and learn how to create containers that can manage transactions and persistence for applications!

Several years ago, it was inconceivable for a developer to write their own enterprise-grade development container or platform. OSGi has drastically changed this; it allows you to create your own domain-specific platform. OSGi is to domain-specific platforms what yacc is to domain-specific languages (DSLs).

Finally, in this book, I tackle some basic problems, but I don't shy away from the complex ones. You'll learn OSGi *in depth*. You won't just be looking at OSGi's API, but rather at the reason why the API is what it is, how you can best use it, and when not to use it. You'll even learn about open OSGi issues and what can be expected to change.

OSGi is an extraordinary technology. More than that, it's a game changer in the way large software is developed. To fully understand the reason for this, you need to understand OSGi *in depth*.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0058

can use a plethora of carrier-grade infrastructure services, such as HTTP, configuration, deployment, event handling, transactions, persistence, RMI, naming and directory services, and management.

We'll start this chapter by exploring development platforms and the benefits of using such platforms to develop software. We'll then discuss the requirements of a platform intended for the development of enterprise-grade applications. Next, we'll focus on the OSGi technology, expanding into its core pieces, called the OSGi framework, and its enterprise services. Finally, you'll learn why OSGi is a good fit as a development platform, particularly in light of existing solutions, such as JEE. Let's start by examining the basics.

1.1 What are development platforms and application frameworks?

In the context of software development, a *development platform* is a set of software libraries and tools that aid in the development of software components, and the corresponding runtime environment that can host these developed components, as shown in figure 1.1.

The *runtime environment* may consist of the hardware, operating system (OS), and supporting runtime libraries. One example of a runtime environment is the Java Runtime Environment (JRE), which includes the Java virtual machine (JVM) that isolates the developer from the details of the underlying OS and hardware.

Software frameworks are specialized types of a development platform's libraries and tools. Wikipedia defines a software framework as an "abstraction providing generic functionality that can be selectively specialized to provide specific functionality."

Particularly interesting to us are application frameworks. An application framework is a type of software framework whose purpose is to provide a structure for the creation of software applications. Applications are programs that allow users to perform related tasks together. Examples of software applications are document editors and antivirus software.

Putting it all together, a development platform allows a developer to create applications and to host these applications so that end users can use them. Throughout this book, it's important to keep these two players in mind: the developer (you) and

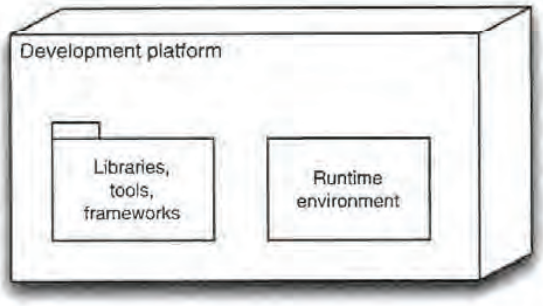


Figure 1.1 A development platform consists of a software framework and its supporting runtime environment.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0059

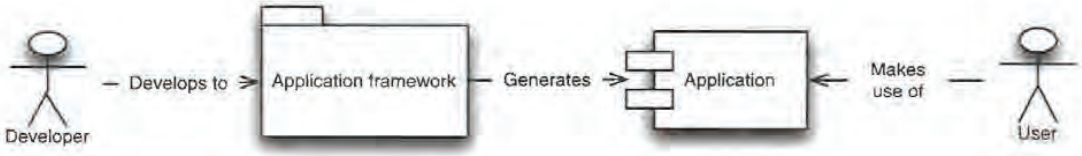


Figure 1.2 A developer creates an application, which is used by a user, and develops to a framework.

the end user, as illustrated in figure 1.2. Software development platforms are also called toolkits or SDKs (software development kits).

Historically, development platforms have always played an import role in software. The Java platform, also known as Java Standard Edition (JSE), is one example. In this case, the Java development kit (JDK) provides the software framework, and the Java Runtime Environment (JRE) provides the runtime environment. The OSGi Service Platform, which is the subject of this book, is another example of a development platform. The OSGi Service Platform uses the JRE as its runtime environment. In addition, it provides an application framework layered on top of the JDK. We'll look into its details throughout this book, but first let's see why it's important to use a development platform to begin with.

1.1.1 Why use a development platform?

Consider the following definition of the word *framework*:

> *"an essential supporting structure of a building, vehicle, or object"*

Why is the supporting structure of a vehicle important? It clearly sounds like it's important, but let's see if I can articulate why that's the case. I can think of two main reasons:

- It guarantees that I'm sitting on top of something that's solid—something that has been designed properly, implemented suitably, and tested thoroughly. A framework helps to decrease defects. This is a runtime characteristic.
- It gives the manufacturer an opportunity to reuse the frame for different vehicles. A framework helps to improve productivity through reuse. This is a design-time characteristic.

It's no different for development platforms. Development frameworks allow the creation of new applications in a form that's both efficient and has a high degree of quality.

1.1.2 Enterprise platforms

Enterprise platforms are development platforms that support the creation of enterprise applications—applications that implement business processes, business logic, or business integration to an enterprise. Examples of enterprise applications are a loan-approval application, an order-processing application, a customer relationship management (CRM) application, and a travel management application.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0060

The following two aspects characterize enterprise platforms:

- Enterprise platforms provide a collection of infrastructure-level utilities and services common to many businesses and industries, such as management, directory service, monitoring, and distribution.
- Enterprise platforms must scale, perform efficiently, and be robust and fault tolerant.

Following up on our previous example, Java also has an enterprise version, which is called Java Enterprise Edition (JEE). Other examples of enterprise platforms are Microsoft's .NET Framework, SpringSource's tc Server Development Edition, and to some extent, Google's Web Toolkit. Recently, a new enterprise platform has been developed, the OSGi Service Platform Enterprise Specification.

As you can see by the number of players, enterprise technology is quite mature, so why is there a need for a new platform, such as the one being provided by OSGi? We'll address this question in section 1.3, but before we can do that, you need to understand OSGi a bit better.

1.2 The OSGi technology

The Open Service Gateway initiative (OSGi) was formed in March 1999 by a consortium of leading technology companies with the mission to define a *universal integration platform for the interoperability of applications and services.*

When I first read their mission, it gave me the impression of being both overly complex and somewhat outdated. Hadn't people already created a universal platform for applications? As you'll learn, no one has been able to do it successfully.

1.2.1 The problem domain

First, let's investigate the underlying problem that these companies were facing. The initial members of the OSGi alliance were in a large part telecommunication equipment manufacturers and service providers. They were interested in deploying software applications on small-memory devices. For example, consider a mobile phone as the device and a location-tracking application and an advertisement application as the software applications being deployed to the mobile phone. The location-tracking application uses the mobile phone to verify the current location of the subscriber and informs the advertisement application of the location. The app then retrieves selected advertisements that are suitable to the current location of the subscriber, such as promotions from nearby restaurants, as shown in figure 1.3.

This seemingly simple interaction caused the equipment and service providers several interesting problems. First, the devices tend to have different hardware and thus different programming APIs. Hence, each vendor had to program its applications to a specific device and then port to other devices. Second, not only do the various hardware devices use different programming APIs, but there's also a large variation in their functions and capabilities. Some have more memory than others; some have a disk whereas others are completely diskless; some have GPS and some do not. Third, the

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0061

There's a clear service contract between the advertisement application and the data source, but the implementation of this contract will vary depending on the device. This universal platform must make it easy for applications to decouple service contracts from the service implementation. The Standard Edition of Java (JSE) doesn't have a service registry or a service management framework that could help us with this. Hence, this facility either needs to be implemented from scratch, or we could try to borrow something from the Enterprise Edition (JEE) of Java. Let's hold on to this thought and tackle it after going through the other problems.

PROBLEM: SUPPORTING DYNAMIC CHANGES

The third problem can be summarized by the following requirement: the platform must allow the dynamic deployment and undeployment of applications in a secure form. Does JSE have support for this? Not really. You could try to solve this with the Java class loaders, but it wouldn't be easy, and there's no simple way of unloading classes after they've been loaded, not to mention that there's no concept of an application deployment unit. The closest concept to an application deployment unit is the idea of JARs (Java Archives), but JARs by themselves don't provide all the metadata that's needed, such as a unique naming schema for the applications.

As with the previous problem, we can implement our own solution for dynamic deployment of applications or try to leverage something from JEE. For example, web servers do have the concept of web applications, which are defined as part of a WAR (Web Archive) deployment unit file.

PROBLEM: PROVIDING A LIGHTWEIGHT SYSTEM

This brings us to the last problem. Whatever solution we pick, it must be lightweight. Yes, we could try to leverage a directory service such as JNDI from JEE or leverage the architecture from web servers, but these solutions would fail to consider the size and memory constraints enforced by the devices onto the platform, making it less suitable for embedded solutions and not a viable option.

1.2.2 The solution: a dynamic module system for Java.

Java addresses some of the problems we've discussed, such as portability, but not all of them. For instance, it lacks proper support for dynamic service management.

Enter the OSGi Service Platform. In its most succinct definition, the OSGi Service Platform, or OSGi platform for short, is a dynamic module system for Java. In OSGi terminology, a Java module is called a *bundle*.

The OSGi Service Platform is composed of two main components, the OSGi framework and the OSGi services, as shown in figure 1.5.

THE OSGI FRAMEWORK

The OSGi framework provides its users with all the pieces that we discussed in the previous section:

- A portable and secure execution environment based on Java
- A service management system, which can be used to register and share services across bundles and decouple service providers from service consumers

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0062

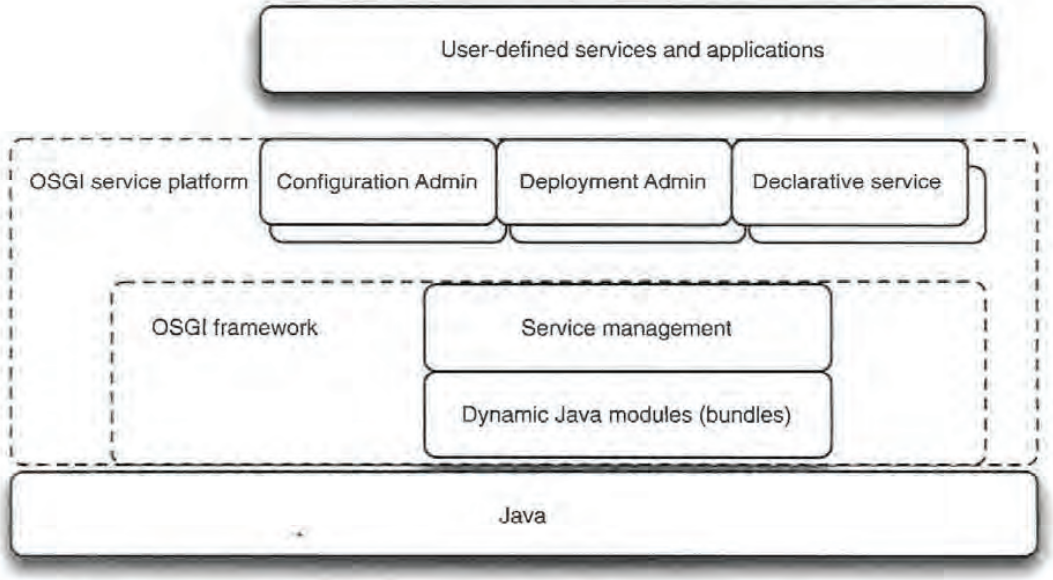


Figure 1.5 The OSGi Service Platform comprises the OSGi framework and the OSGI services.

- A dynamic module system, which can be used to dynamically install and uninstall Java modules, which OSGi calls bundles
- A lightweight and scalable solution

The OSGi framework is the core structure of the OSGi Service Platform. It can be seen as a backplane that hosts bundles, possibly containing services. If you consider that a bundle can be an application, then the definition of the OSGi framework is in accordance with our definition of application frameworks. That is to say, the OSGi framework is an example of an application framework.

Right now, we'll leave the definition of the OSGi framework somewhat loose. Let's not worry about what exactly bundles and services are. We'll discuss the framework, its concepts, and its APIs in detail in the next chapters.

THE OSGI SERVICES

Alongside the OSGI framework, the OSGI Service Platform includes several general-purpose services. You can think of these services as native applications of the OSGi Service Platform.

Some of these services are horizontal functions that are mostly always needed, such as a logging service and a configuration service.

Some are protocol related, such as an HTTP service, which could be used by a web-based application.

And finally, some services are intrinsically tied to the framework, which won't work without them. Examples of these are the bundle wiring, which manages the dynamic module system itself, and the start-level service, which manages the bootstrap process of the framework.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0063

The focus of the initial releases of the OSGi Service Platform had been on the OSGi framework, but gradually we see the OSGi services playing a more prominent role. This trend toward other components, such as services, built on top of the core OSGi framework is a reflection of the increasing popularity of the technology.

1.2.3 The Enterprise OSGi

As can be deduced from its history, OSGi was initially employed in the embedded market. But with its growing popularity and maturity, OSGi is moving to the enterprise market. To address this requirement, the OSGi Enterprise Expert Group (EEG) created the OSGi Service Platform Enterprise Specification (Enterprise OSGi).

This specification combines OSGi services that can selectively be used to provide enterprise functionality. These services can be grouped into enterprise features. Examples of enterprise features, as shown in figure 1.6, are the following:

- *Management and configuration*—This group includes the Configuration Admin service as well as the JMX Management Model service and the Metatype service.
- *Distribution*—This feature allows the communication of end points between remote instances of OSGi framework instances. Some of the services in this group are the Remote Service and the SCA Configuration Type.
- *Data access*—This feature includes support for JDBC, JPA, and JTA and allows the manipulation of persisted objects.

Before getting into the details of the OSGi technology itself, you should understand the high-level benefits of using OSGi as your development platform. We address this topic in the next section.

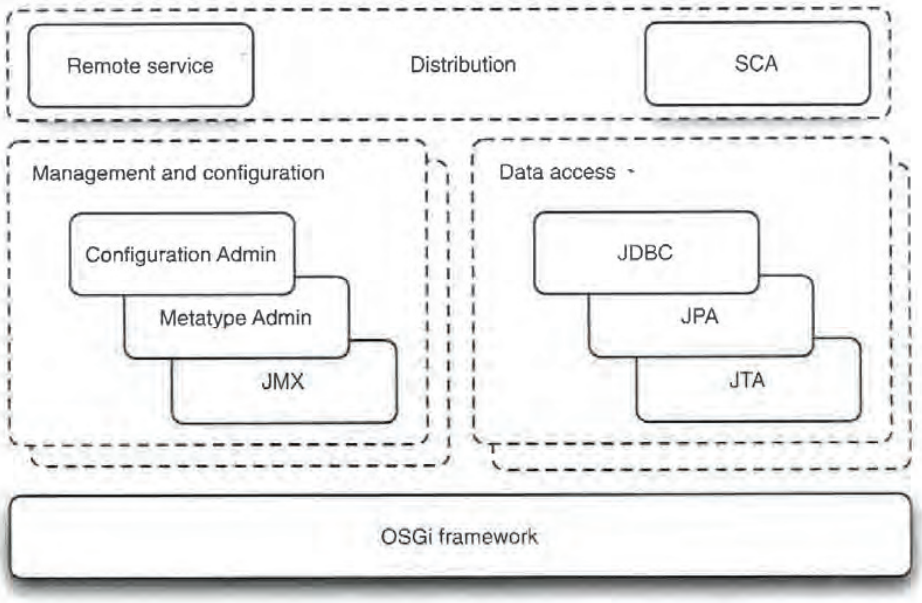


Figure 1.6 Enterprise OSGi consists of the OSGi framework and several OSGi services; together they provide enterprise features.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0064

structure of a program in a productive and efficient form. You may point out that the concept of modularization has been used in software engineering for decades, so why is this any different now? The difference is that OSGi allows you to apply modularization in a systematic form to software systems at their very foundation, which was never attempted before.

1.7 Summary

Development platforms consist of an application framework and a supporting runtime environment. The Java platform, with the JDK and JRE, is one example of a development platform. Enterprise platforms, such as Java Enterprise Edition (JEE), add enterprise features to the platform.

The OSGi Service Platform provides a dynamic Java module system for Java. It allows Java code to be modularized and to be managed as services. The OSGi Service Platform consists of the OSGi framework and the OSGi services. The OSGi Service Platform Enterprise Specification (Enterprise OSGi) was created to support enterprise use cases. It defines a collection of OSGi services that can be used together for enterprise features.

OSGi provides the means to achieve modularization, which helps decrease and manage the complexity of large systems. OSGi provides a native extensibility mechanism through the use of services. Finally, a consortium of several large companies is driving the OSGi specifications, hence aiding with its adoption as a standard.

In the next chapter, we'll drill down into the OSGi framework in detail. In chapter 3, we'll employ what you've learned so far to build your first OSGi-powered application. But this application will lack several useful features, such as persistence. Following up on this, we'll take an in-depth look into how we can use configuration, event handling, persistence, RMI, transactions, naming and directory services, and management to develop full-fledged carrier-grade OSGi solutions.

Finally, you must keep in mind that the main purpose of OSGi is to provide you with an efficient framework for creating and integrating software building blocks.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0065

But there are cases when this doesn't apply. For example, consider the case where a single bundle wants to open different Telnet sessions. This isn't possible today, because all the `TelnetService` services retrieved by this bundle would point to a single `TelnetImpl` object instance. In my opinion, this behavior should be made configurable, perhaps through the specification of some new predefined service property, so as to follow the same pattern initiated by `SERVICE_RANKING` and the other predefined properties. In the end, this is a result of overloading the function `getService()` to both indicate the retrieval of a service as well as the creation of a service when using a `ServiceFactory`. Therefore, perhaps another option would have been to define a `createService()` method in the `BundleContext`.

We've looked into several advanced features of OSGi services that help us improve decoupling. Next, you'll learn how to test your applications.

4.7 Improve robustness by testing your applications

In the previous sections, we went through several issues pertaining to developing OSGi bundles and services. One final issue related to the process of developing OSGi applications remains: how do you test OSGi bundles effectively?

4.7.1 Unit tests

It's a good idea to start by testing a bundle at the unit level. You want to make sure that the bundle, which generally is defined by a well-established contract or interfaces, works in isolation. For a standard library or module, this is generally done by creating JUnit test cases. The good news is that you can continue using JUnit to test your OSGi bundle. This is so because OSGi avoids imposing any of its technology-specific interfaces on the application. This allows developers to test their classes as they normally would.

Let's look into an actual example. Recall our auction framework from chapter 3? How could we test the `SealedFirstPriceAuction` class of the `auction.auctioneer.sealed` bundle, which is where the most complex application logic resides in the case of the auction application?

Because we were careful to isolate the OSGi-related code to a single class called `SealedFirstPriceAuctioneerActivator`, you can instantiate the class `SealedFirstPriceAuction` in isolation within a JUnit `Testcase` class without having to worry about the OSGi environment, as shown in the following listing.

Listing 4.9 A unit test for the `SealedFirstPriceAuction` class

```
public class TestSealedFirstPriceAuction extends TestCase {

    static class MockParticipant implements Participant {

        int numberOfAcceptances = 0;

        public String getName() {
            return "participant";
        }
```

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0066

```
        public void onAccepted(Auction auction, String item, Float price) {
            numberOfAcceptances+=1;                          ❶ Count acceptances
        }

        public void onRejected(Auction auction, String item, Float bestBid)
        {
            fail("Rejected");
        }
    }

    public void testSingleBidAsk() throws InvalidOfferException {

        SealedFirstPriceAuction auction =
            new SealedFirstPriceAuction(1);          ❷ Class to be tested

        MockParticipant participant =
            new MockParticipant();                   ❸ Mock class

        auction.ask("book", new Float(50.0), participant);
        auction.bid("book", new Float(50.0), participant);

        assertEquals(2, participant.numberOfAcceptances);   ❹ Check results
    }
}
```

In ❷, you instantiate the test subject as a regular Java class, which in fact it is. You create a mock participant ❸, which only counts the number of accepted callbacks that were invoked ❶. Finally, you test the auction by sending a single bid and ask that match, and you assert the results ❹.

To run this test, which you can do through the command line or in some IDE such as Eclipse, you need to make sure the following JARs are in the class path: the JUnit library, the auction bundle, and the OSGi framework API (in our case, felix.jar will do). The latter is needed only because of the `SealedFirstPriceAuctioneerActivator` class. Remember that our test is a JUnit test case, so the easiest option is to run it using a JUnit runner, which is provided in the JUnit library itself.

What if we hadn't been so careful and had polluted our application logic with OSGi-related interfaces? Or what if you just wanted to unit test the bundle activator class itself?

In this case, there are two options: you could either run the unit test within the OSGi platform itself, or you could emulate (that is, mock) an OSGi environment for the unit test. Fortunately, the Spring Framework provides an OSGi mock library, making the latter approach an attractive one. Continuing with our example, let's unit test the `SealedFirstPriceAuctioneerActivator` class, as shown in the following listing.

Listing 4.10 Unit test for `SealedFirstPriceAuctioneerActivator` class

```
public void testAuctionnerActivator() throws Exception {

    MockBundleContext mockContext = new
        MockBundleContext();                    ❶ Mock BundleContext
```

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0067

```
SealedFirstPriceAuctioneerActivator activator =
    new SealedFirstPriceAuctioneerActivator();

activator.start(mockContext);

ServiceReference reference =
    mockContext.getServiceReference(Auctioneer.class.getName());

assertNotNull(reference);
}
```

❷ Activator to be tested

❸ Start using mock context

❹ Validate activator

In ❷, you create a bundle activator instance outside the OSGi environment. Then you create Spring's mock bundle context ❶ and use it to start the activator ❸, as if the activator was being started by the OSGi platform. Then you validate the activator by making sure that it has registered a service reference for the auction service ❹.

The Spring Framework

The Spring Framework is known for the popular dependency-injection library provided by the company SpringSource, among other things. SpringSource has adopted the OSGi platform and is providing several tools, libraries, and its own OSGi application server!

You can also run this test in isolation similarly to the previous case, but you must remember to add the Spring OSGi mock library to the class path.

NOTE Spring OSGi libraries, also called Spring DM, can be found at http://www.springsource.org/osgi.

After having unit tested all of our bundles, the next step is to make sure the bundles are able to talk to each other within the OSGi platform itself. This process is called *integration testing*.

4.7.2 Integration tests

To execute integration tests for our bundles, you'll need to run the bundles and the tests within an actual OSGi platform. And obviously, because you want to automate the execution of the tests, you'll want to do this through scripts or the command line. In essence, you need to do the following:

1. Start the OSGi platform.
2. Install all the bundles that need to be tested and their dependencies.
3. Install the test itself; this can be in the form of another bundle.
4. Trigger the execution of the test by invoking an MBean from a script or by having the test bundle include a bundle activator that starts the execution of the test.
5. Store the verdict (that is, pass or fail) of the test; writing the result to a file can do this.
6. Shut down the OSGi platform, and repeat the process if other tests are needed

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0068

Fortunately, Spring DM also provides a facility for helping you doing this. Simply have your test extend the JUnit abstract class `AbstractConfigurableBundleCreatorTests`. This base class will take care of starting the OSGi platform, creating a test bundle, executing it, and shutting down the platform, as shown in the following listing.

Listing 4.11 Integration test for the auction framework

```
import org.osgi.framework.ServiceReference;
import org.springframework.osgi.test.AbstractConfigurableBundleCreatorTests;
import org.springframework.osgi.test.platform.Platforms;

import manning.osgi.auction.spi.Auctioneer;

public class IntegrationTestAuction extends
        AbstractConfigurableBundleCreatorTests {

    public void testSealedFirstPriceAuction() {
        ServiceReference reference =
            bundleContext.getServiceReference(
Auctioneer.class.getName());

        Auction auction =
            (Auction) bundleContext.getService(reference);

        // Place bids, asks...
    }

    protected String getPlatformName() {
        return Platforms.FELIX;
    }
}
```

❶ Extend test framework class

❷ Use service as usual

❸ Specify OSGi framework

To test the auction framework, you create a JUnit class `IntegrationTestAuction`, which extends the Spring DM helper class `AbstractConfigurableBundleCreatorTests` ❶. This class is run as a bundle, so it has access to a `bundleContext`. With the `bundleContext`, you retrieve the Auction service ❷ and perform the actual test by placing bids and asks as you did previously, but now the whole interaction is done through the OSGi framework.

You can test the JUnit test case using a JUnit runner. But you need to include several JARs in its class path this time: the JUnit library, the Spring DM libraries and their dependencies, and the OSGi platform libraries, which in our case is felix.jar. You should run the test from the Felix home directory, so that the test helper class is able to start Felix correctly. You also need to tell the helper class that you want to use Felix and not some other OSGi platform, such as Equinox. This is done at ❸. And finally, don't forget to install all the necessary bundles, such as all of the auction application bundles, in Felix before running the test; otherwise they won't be found. It still seems a fair amount of work, but it's simpler than doing all these tasks manually.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0069

4.8 Summary

In this chapter, we explored several advanced OSGi bundle and service features that allow you to develop flexible, extensible, and robust OSGi applications.

The OSGi framework provides several options for minimizing bundle dependencies. You can restrict a resource to be within a private package and thus not visible to any other bundle. You can also choose to export only a subset of the classes residing in a package, thus reducing the number of classes that make up the exported package contract. Finally, you learned that you should avoid creating bundle dependencies using the `Bundle-Required` header, because it can cause split packages and cause other problems.

We looked into three OSGi features that facilitate the creation of generic bundles. Dynamic imports provide a way out of the explicit import and export package contract, because they allow classes to be searched as needed. Optional packages are regular import packages that aren't considered an error if they're not resolved. And finally, fragment bundles are degenerated bundles that attach and merge with a host bundle, possibly extending its class path.

When a bundle loads a resource, this resource is searched through the bundle's class space, starting at the boot class loader, then at any `Import-Package` header, next at the bundle's private package, and finally at any fragment bundles attached to the bundle.

You learned how to specify complex service reference filters. OSGi service filters follow a prefix form, which can be non-intuitive but allows several operations to be grouped together on a single conjunction or disjunction.

You can use the predefined `SERVICE_RANKING` property to prioritize which service is used if multiple services meet a criterion when being retrieved from the OSGi service registry. This is useful when you need to specify which service among many should be the default one.

The predefined `SERVICE_PID` property is used to uniquely identify a service. This is useful when a service represents a real-world entity. No two services should be registered with the same `SERVICE_PID`, and the `SERVICE_PID` must be persisted; that is, the same value must be used when a service is restarted.

Generally, a service reference maps to a single service implementation object instance. But if each service must have its own state, then multiple service implementation object instances must be created for the same service reference. To do this, a service implementation object must implement the interface `ServiceFactory`.

The OSGi framework imposes very few vendor interfaces into the application classes, so it's generally easy to unit test a bundle by treating it as a regular Java class. Spring DM provides several mock classes for the OSGi framework API. Spring DM also allows you to perform integration tests. It has helper classes that start an OSGi platform, install bundles, and allow you to test their integration.

In the next chapter, we'll look at how to configure these decoupled bundles in a way that's consistent for the entire application.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0070

10 Remote services and the cloud

This chapter covers

- Importing and exporting a remote OSGi service
- Understanding distribution providers and their properties
- Selecting a specific service endpoint and negotiating endpoint policies
- Learning the semantics of dealing with remote entities
- Understanding cloud computing and how it relates to OSGi

In today's world, it's almost guaranteed that your machine is networked and the services you use are provided through the internet. Remote communication is essential, and OSGi supports it by means of the OSGi Remote Service specification, which is the subject of this chapter.

We'll start by examining how to export a service for remote access and conversely how to import a remote service for local consumption. Next, we'll investigate how

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0071

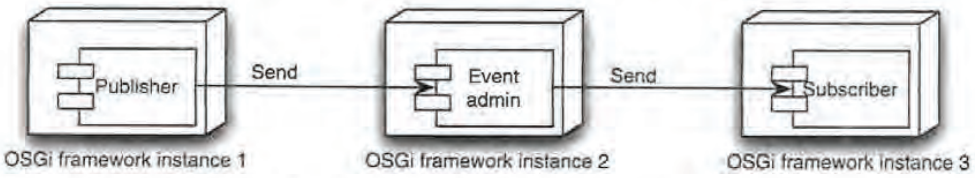


Figure 10.1 The publisher, subscriber, and Event Admin communicate remotely.

the actual transport is realized through the use of distribution providers, highlighting the fact that OSGi doesn't implement its own transport but rather allows existing transports to be plugged in. Finally, we'll look into how to negotiate common policies between the exporter and the importer of the remote services, for instance, to make sure that both sides agree on a common set of requirements, such as encryption.

Remote services opens up OSGi to be run in a remote server or, as is the tendency, in the cloud. We'll briefly discuss cloud computing and look at how the OSGi platform is an ideal platform for cloud computing. But before getting into cloud computing, we'll start the chapter by discussing remote service invocation.

10.1 Remote invocation

Recall from chapter 6 that one of the shortcomings of the Event Admin service is that it doesn't handle remote clients. Ideally, subscribers that have registered the `EventHandler` interface should be invoked regardless of their location.

For example, you could have a scenario where the publisher, the actual Event Admin implementation, and the subscriber are hosted in three different OSGi framework instances, as depicted in figure 10.1.

This distributed configuration can be achieved using OSGi's Remote Service. In this setup, distribution providers are responsible for creating proxies that represent remote services, therefore achieving a distributed system. Let's investigate the pieces, starting with a remote `EventHandler` subscriber in the next section.

10.1.1 Exporting a remote service

A service is specified as remote by tagging it with the following service properties:

```
service.exported.interfaces
service.exported.configs
```

The former specifies which service interfaces are to be exported in a remote fashion, and the latter specifies how those interfaces are to be made remote. The following listing provides an example for this remote subscriber scenario.

Listing 10.1 Remote `EventHandler` service

```
Dictionary props = new Hashtable();

props.put("service.exported.interfaces", "*");   <-- (1) Export EventHandler interface as remote
props.put("service.exported.configs",
```

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0072

invocation might unexpectedly fail or yield a high latency because of problems with the underlying transport.

Is there a way of knowing if a service is remote? Yes, all remote services have the following service property:

```
service.imported
```

This property is useful for filtering. For example, the following expression filters out all remote services, making sure that only local services are used:

```
(!(service.imported=*)
```

Imported services are also set with the service property `service.imported.configs`, which specifies the configuration type used for the particular endpoint of the imported service. For example, in our case it would be set to `org.apache.cxf.ws`. We'll discuss configuration types in depth later.

Finally, keep in mind that imported services are *not* set with the `service.exported.*` properties, such as `service.exported.interfaces`.

Next, let's take a quick look at how things actually work under the covers.

10.2 Distribution providers

It's the role of distribution providers to implement the actual transport between the bundle that's exporting a remote service in one OSGi framework instance and the bundle that's importing the service in another framework instance. A distribution provider can support different transports, such as RMI (Remote Method Invocation), SOAP (Simple Object Access Protocol), and REST (Representational State Transfer). The transport to be used is determined by the configuration type specified in the `service.exported.configs` service property. An endpoint description is created for each remote service and a particular configuration type. Consuming bundles use the endpoint description to establish a connection to the remote service.

> **NOTE** The OSGi Remote Service specification is a framework for remote communication that supports different transports, such as RMI and SOAP. The specification itself doesn't specify which transports are supported. Further, it doesn't define any new transport protocols.

A remote service may have multiple endpoints. For example, a remote service may be exported with both an RMI and SOAP endpoint descriptions. Clients can then select the proper endpoint for invocation. This approach allows the support of different transports, in a manner agnostic to the remote service and its consumers. Figure 10.2 describes the interaction between distribution providers, their supported configuration types, and the endpoints created by each configuration type for each remote service.

As you can infer, a distribution provider is needed for both the consuming and the providing sides, as shown in figure 10.3. On the consuming side, the distribution provider takes care of importing the remote service by establishing a local proxy endpoint

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0073

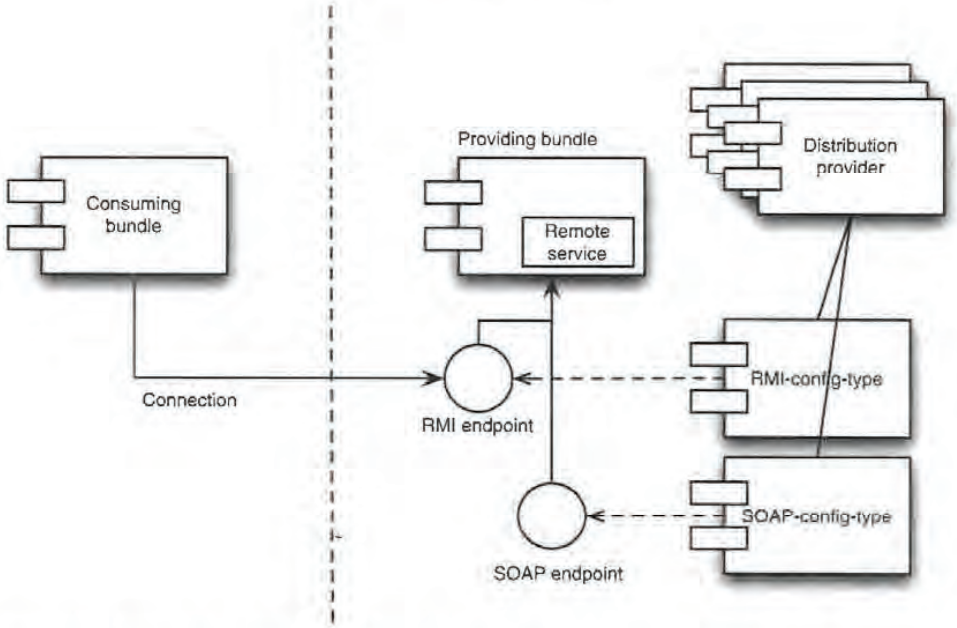


Figure 10.2 Distribution providers support different configuration types, which create separate endpoints for each remote service. In this example, the provider supports RMI and SOAP transports. Each transport endpoint is associated with the remote service and is tied to the client when a connection is established.

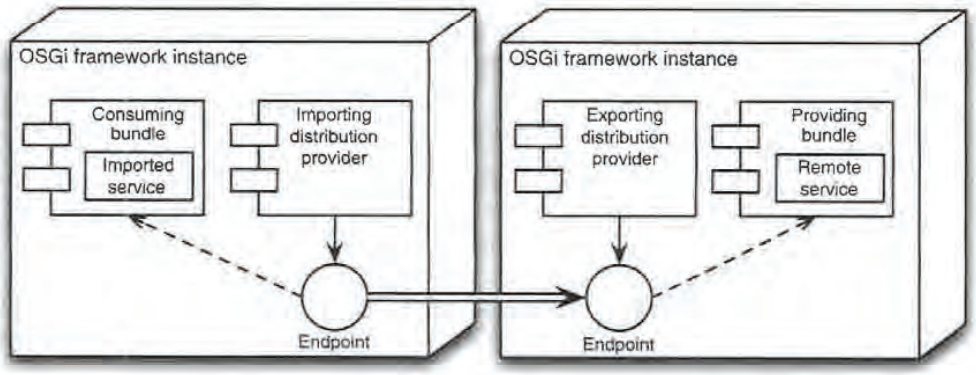


Figure 10.3 There's a distribution provider in each distributed framework instance, each responsible for its local endpoint and connecting to a remote endpoint.

and connecting it to the remote endpoint. We'll call it the *importing distribution provider*. On the providing side, the *exporting distribution provider* is responsible for establishing an endpoint that accepts connections and delegates to the proper service.

The distribution providers are also responsible for discovering and proxying the remote services as needed. That's why you don't need to explicitly mention the address of a remote service in the consuming bundle.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0074

You've learned how a distribution provider works. In the next section, we'll look at how to handle multiple distribution providers and transports.

10.2.1 Selecting the proper endpoint

It's not uncommon for a system to support multiple transports. For example, RMI for general application communication, SOAP for web development, and perhaps some proprietary socket implementation (perhaps running on InfiniBand) when performance is essential.

When exporting a remote service, you should generally be as flexible as possible and therefore specify several or the entire set of supported configuration types. For example, we could export the `EventHandler` setting the `service.exported.configs` service property to `net.rmi, net.soap, net.socket`. If we did so, and the distribution provider supported all the aforementioned transports, then three distinct endpoints would be created, one for each transport.

But on the consuming side, we want to import the service using a single specific endpoint, as the situation demands. For example, if we wanted to use the event handler interface from a web application, then it's likely that we'd prefer to import the `EventHandler` service using the SOAP endpoint. This can be done using the `service.imported.configs` service property. The following code fragment retrieves an `EventAdmin` service that's tied to the SOAP endpoint:

```
ServiceReference ref =
   context.getServiceReference(EventAdmin.class.getName(),
      "(&(service.imported.configs=net.soap))");
```

Unlike the `service.exported.configs` property, the `service.imported.configs` property contains the single configuration type that's being used by the supporting endpoint, as shown in figure 10.4.

If the importing distribution provider in the OSGi framework supports all three transports we specified when exporting the remote service, then three endpoints are created and likewise three services are imported, each containing a different configuration type for its `service.imported.configs` property.

> **NOTE** The fact that endpoints for all configuration types are created may seem wasteful at first glance, but in reality a distribution provider implementation may do this on demand by using the OSGi service hooks.

But what happens if the importing distribution provider doesn't support the transport we specified? In other words, in the preceding example, how would we know there's a SOAP endpoint for us to depend upon? We can find the supported configuration types of a distribution provider by using the service property `remote.configs.supported`. The distribution providers register some anonymous service, setting this property to the list of supported configuration properties. In our case, should the importing distribution provider only support the configuration types of `net.rmi` and `net.soap`, then it would register a service whose `remote.configs.supported` service property is set to `net.rmi, net.soap`.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0075

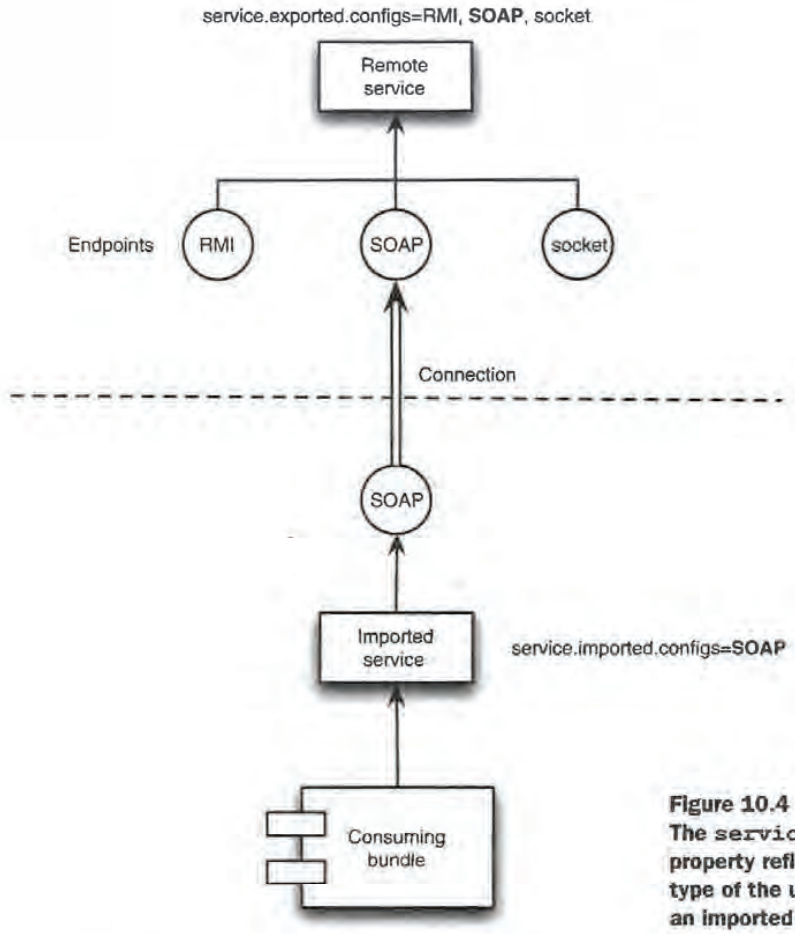


Figure 10.4
The `service.imported.configs` property reflects the configuration type of the underlying endpoint for an imported service.

Knowing this, our consuming bundle should first find out if the `net.soap` configuration type is supported and only then retrieve the SOAP endpoint for the `EventHandler` service. We can find out if the `net.soap` configuration type is supported as follows:

```
ServiceReference ref =
   context.getServiceReference(null,
      "(&(remote.configs.supported=net.soap))");
```

Finding out the supported configuration types is useful, but sometimes you may need more information, such as whether a particular policy is supported. This is the subject of the next section.

10.2.2 Negotiating policies

There are cases when not only do you need a particular endpoint but you also need to certify that a policy from the remote service or distribution provider is being supported. It's sometimes even necessary to negotiate policies between the consuming and the providing systems and make sure that both sides of the equation agree on common vocabularies.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0076

`setAddress()` method from the `Customer` object, which mostly becomes a structure of values. Because the consuming bundle generally wouldn't know it's dealing with a remote service, the onus of the interface design falls largely on the provider.

The next problem we'll look at is that a communication transport may go down for a variety of reasons at any arbitrary time. There isn't much we can do to deal with this in the provider, and instead we need to design for this possibility in the client itself. For instance, just before we invoke the `updateCustomerAddress()`, our wired or wireless network connection may be lost, and the importing distribution provider would lose connectivity to the exporting distribution provider. In this case, the imported service should raise the runtime exception `ServiceException`, and a consuming bundle should be ready to handle this exception.

A naïve exception-handling implementation for a `ServiceException` might wait a few seconds in case the network has simply run into some transient problem and try again, as demonstrated in the following code:

```
ServiceReference servRef =
    bundleContext.getServiceReference(RemoteCustomerRegistry.class.toString());

if (servRef != null) {
    RemoteCustomerRegistry registry =
        (RemoteCustomerRegistry) bundleContext.getService(servRef);

    try {
        registry.updateCustomerAddress("Alex", "Updated Address!");
    } catch (ServiceException e) {
        Thread.sleep(2000); // wait for two seconds and try again

        registry.updateCustomerAddress("Alex", "Updated Address!");
    }
}
```

One final issue to consider is the underlying communication stack being used. The different transports have different characteristics and limitations. For example, if CORBA is being used, there are restrictions on the Java types that can be referenced in the remote services. This is because CORBA IDL (Interface Definition Language) doesn't support the entire spectrum of the Java types.

Remote services, sometimes called Distributed OSGi, are a major foundation for the emerging cloud technology, which we'll investigate in the next section.

10.4 Elasticity at the cloud

Needless to say, cloud computing is a vast subject, and it's beyond the scope of this book. But as you'll see, OSGi has become an interesting technology for supporting the concept of Platform as a Service (PaaS). In the following sections, we'll make some assumptions and simplifications as we discuss why and how that is.

Let's begin by exploring a use case. Suppose we'd like to make the auction application, which was explained in chapter 3, a public service for a website. Somewhat similar to eBay, the goal is to support web clients making offers of items to sell and bidding

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0077

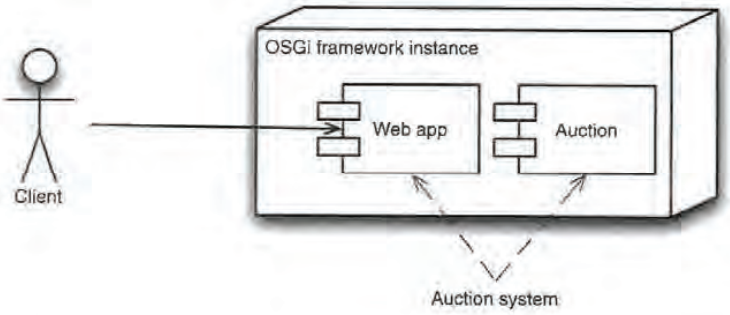


Figure 10.8 A single OSGi framework instance running the auction system

for these items. The first step toward achieving this is to create a web application that exposes a proper web page, where a customer makes offers and bids.

Simply put, the web application internals implement the `Participant` class and invoke the `Auction` class. And we are, in a naïve sense, finished. The web application is implemented as an OSGi bundle, and together with the auction application bundles and the bundle containing the web server (for example, Jetty), they're all installed in a single OSGi framework instance and hosted in a machine that's made accessible via the net. We'll refer to this whole environment as the *auction system*. That is, the auction system includes all the software components needed to run the auction application, such as the web server and any other supporting components, as shown in figure 10.8.

Life goes on; the auction system is running successfully and in fact is a major success—so much so that we get more customers than we expected, and because of that, performance degrades significantly. Whereas before buyers and sellers were able to bid and make offers instantly, it now takes several seconds, sometimes minutes if the item is very popular. The solution is to expand and acquire another machine. This trend continues, and we reach the point of having to manage a network of eight machines. This setup has become extremely expensive; not only do we need to buy several fully configured machines, but we also need to manage them. We decide to do some profiling to look for the bottleneckand find out what's causing the performance problems.

10.4.1 Designing for the cloud

As it turns out, the bottleneck occurs when correlating the offers and the bids in the `Auction` bundle implementation. Although we could try to improve the algorithm, this already tells us that the bottleneck isn't I/O bound and likewise isn't in the web application.

We decide to change our architecture. Rather then having eight instances of the OSGi framework each running the auction system, which consists of the web application, the auction application bundles, and the supporting infrastructure bundles, we change it to a two-tier architecture, as shown in figure 10.9. In our two-tier architecture, the first tier consists of the web application and the web server, which provides the client interface to the customers and then forwards the requests to the second tier,

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0078

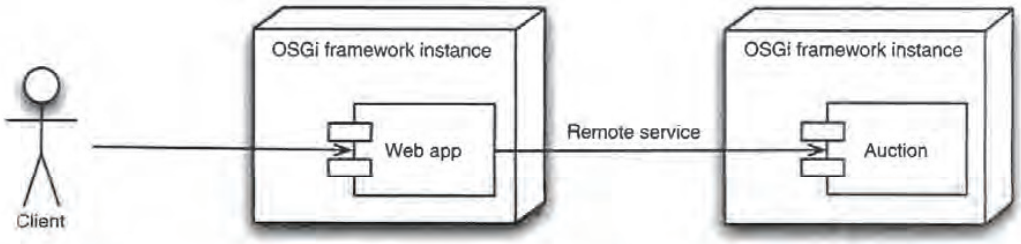


Figure 10.9 An auction system partitioned into two separate OSGi framework instances, communicating through an OSGi remote service

called the business tier. The business tier contains the auction bundles, because they perform the actual business-related code.

In this new setup, we host the web applications in only two machines, which communicate via REST with four other machines containing the auction application bundles. Because of the performance improvements, we're able to serve the same number of clients with only six machines, two fewer than before.

> **NOTE** In reality, we should have a three-tier architecture. The auction application would store its state in the third tier, the data tier. You'll see how to change the auction application to persist its data by using JPA in a future chapter. For the time being, we'll ignore this issue and focus on the communication aspect of the system rather than the persistence issues. But keep in mind that to function properly in the absence of a data tier, we'd need to make sure that all requests pertaining to an item are routed to a single auction bundle instance.

By doing this, we gain on several fronts:

- The machines are easier to manage. When a web server security patch is released, we only need to update the two machines of the client tier. Likewise, we only need to set up the firewall in these two machines. Further, had we been persisting the data, we'd only need to install the persistence storage, such as the RDBMS, in the machines of the data tier.
- The overall environment is less expensive, because we can customize the machines for their specific roles. For example, the machines in the client tier need better I/O but don't need particularly faster CPUs or disks. But the machines in the business tier should have fast CPUs, because we know from the profile that the auction application is CPU bound. By segregating the system into separate nodes, we can avoid having to buy a machine that has everything, such as fast I/O, fast CPUs, and fast disks. Instead, we can be smart and use cheaper machines without impacting the result.
- Performance improves, because we allow the machines that had the bottleneck to focus on just one thing, correlating the bids and offers, rather than also spending CPU cycles on the web application.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0079

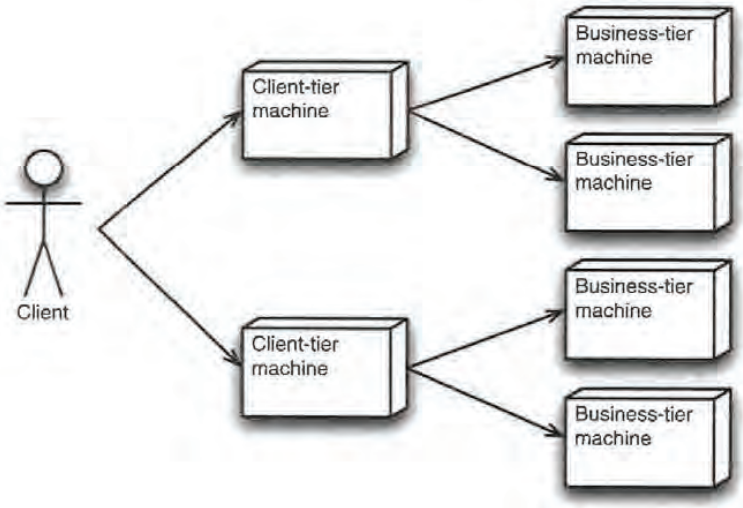


Figure 10.10 Two-tier architecture for the auction system

- The system is overall more scalable, because we have a flexible setup, where we understand which components can be replicated and how best to partition the requests.

Surprisingly, it's quite easy to change our existing implementation to the new two-tier architecture. Instead of collocating all the bundles of the auction system in a single OSGi framework instance, we partition the system into several separate OSGi framework instances, placing the web application bundle in the OSGi framework instances hosted in the client tier machines and placing the auction application bundles in the business tier machines, as shown in figure 10.10. The main effort is in making sure that the web application bundle is able to locate and use the remote Auction service. This, as you've seen in this chapter, is done by using distribution providers and setting the proper service properties, such as `service.exported.configs`.

During this exercise, you've learned an important albeit subtle lesson. The reason we've been able to easily move from a local environment to a multiple-machine environment and finally to a two-tier multiple-machine environment is that our auction system is modular. Had we designed the whole system as a single monolithic (web) application, we wouldn't have been able to partition it when needed. As you'll see next, being able to partition our system is a prerequisite to running in the cloud.

Let's continue our story. We've been running our auction system successfully for several months now. During this period, we've noticed that business is very slow on some dates, like at the end of January, and business is very good in other cases, like near Valentine's Day. This burst-like behavior is difficult to manage and eventually becomes a problem. If the business continues doing well, we may have to double the number of machines we have for Christmas, only to have them sit idle in the months that follow. We need to be able to dynamically cope with this variation of demand. Coping with this dynamism is the exact proposition of cloud computing, as you'll see next.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0080

10.4.2 Cloud computing

The National Institute of Standards and Technology (NIST) defines cloud computing as follows:

> **DEFINITION** Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.

To utilize cloud computing, we need to move our auction system from being hosted on our own machines to being hosted in machines managed by a cloud provider, as shown in figure 10.11. In fact, we don't even need to know the physical layout and characteristics of these machines; they're simply computing nodes performing their work. To do this, we create a virtual machine (VM) containing all of our software components—the operating system (OS), generally Linux, the OSGi framework distribution, and all of the application bundles and supporting libraries, like the Jetty web server. Our VM not only contains the binaries but also knows how to start them appropriately. For example, the VM should bootstrap the OSGi framework instance and make sure that our bundles have been installed and are activated. We can do this by setting up a shell script as an OS service. OS services are automatically started when the VM and thus the OS is booted.

Next, we give the VM to the cloud provider and tell the provider to assign six computing nodes to the VM. That is, our VM is deployed in six nodes somewhere in the cloud. As it gets close to Christmas, we ask the cloud provider to assign four additional

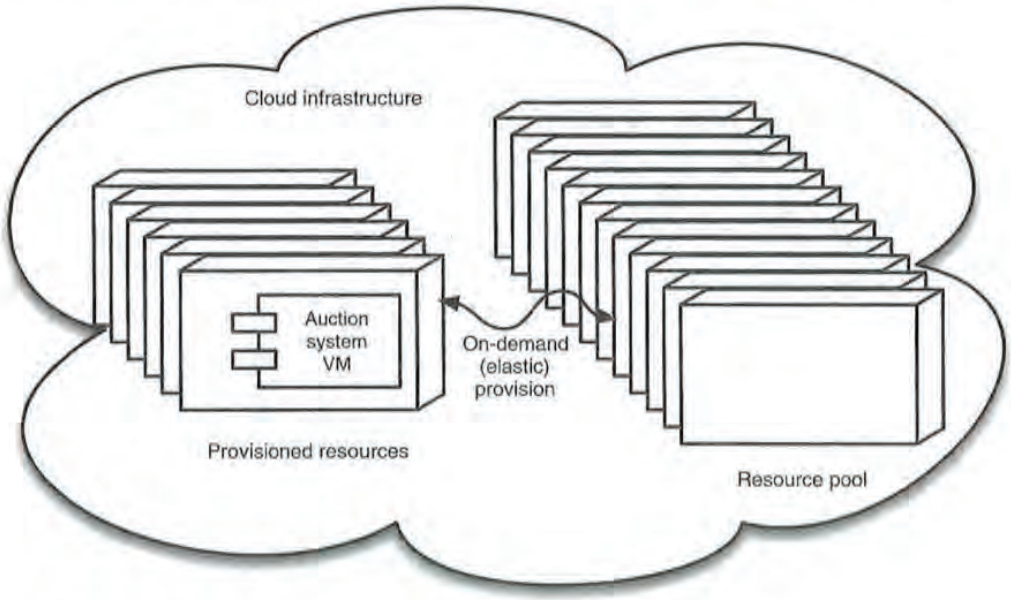


Figure 10.11 Auction system running in the cloud

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0081

nodes to our VM. After the holiday season ends, we ask the cloud provider to remove the four additional nodes; in fact, we ask it to keep only four nodes total running our VM. The cloud provider has a pool of resources, these being computing nodes, disks, and the like, which can be dynamically provisioned to run the customer's software.

> **DEFINITION** The cloud is made up of a pool of resources that are provisioned dynamically on behalf of the customers.

In our particular case, we're assigning the nodes per calendar date, but we could even be more ambitious and monitor the utilization of the resources. For example, we could set up provisioning rules, as follows:

- If the CPU crosses an 80% utilization threshold, then automatically provision a new node for our VM.
- If the CPU utilization falls to below 20% for more than two days, then automatically shut down the node.

As you can see, cloud computing allows us to dynamically and efficiently change our environment based on demand. It also has other advantages, just like any other hosted service, such as us not having to manage our own machines and networks. But keep in mind that the main proposition for cloud computing is indeed its elasticity.

> **WARNING** Cloud computing also has its drawbacks, one of the most serious being security. Because services from different customers are all sharing resources, there's always a higher likelihood of security breaches. Another disadvantage of cloud computing is the increased latency because services may hop several times between computing nodes before being fully served.

The question that remains to be answered is how OSGi relates to cloud computing. We'll address this question in the next section.

10.4.3 OSGi as the cloud platform

Let's say that to keep our business competitive, we need to fine-tune our auction algorithm frequently. Updating our auction system in the cloud isn't a trivial task; we need to change the auction bundle, re-create a new VM image, upload it to the cloud, shut down the VM instances, and finally replace them with the new image.

Generating VM images and the whole workflow process just described can be quite costly, especially if it must be done on a daily basis. Moreover, keep in mind that when generating a new VM image, we need to test all the components, and the time used for testing is proportional to the number of components being tested. By using OSGi, we can modularize our systems and therefore partition adequately, creating simpler and smaller VMs. You already saw an example of this when we partitioned the auction system into two tiers. The result is that we can have two separate VM images, one containing the components of the client tier and the second containing the components of the business tier. Managing smaller VMs is obviously better. For example, if we're updating just the auction bundle, it would prevent us from having to repackage and

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0082

retest the web application and server. Potentially, we could also rent cheaper computing nodes for the client tier VM, which contains the less-demanding web application bundles.

> **WARNING** It's a common misconception that you don't need to design your system to run in the cloud. To fully utilize cloud computing, the system must be able to scale. Modular systems scale better; therefore, OSGi is a clear infrastructure choice, because it emphasizes and helps in the creation of modular applications.

That's not the complete story though. Suppose that instead of uploading and managing VMs in the cloud, we were allowed to manage OSGi framework instances and their bundles. If this were the case, we'd just need to inform the cloud provider to update our auction bundle. The auction application, being an OSGi application, already has to deal with the fact that dynamic changes and updates may happen, so the fact that it's happening when it's running in the cloud or running as a standalone local application is mostly immaterial. Thus, if our supposition were true, OSGi would give us two important advantages:

- OSGi would let us update our system quite easily and efficiently, without us having to get into the business of generating full VM images.
- It avoids lock-ins, because we can design and validate our applications using any OSGi implementation in a standalone environment before moving to the cloud.

We could even take it a step further. The cloud provider could give us access to an OSGi bundle repository (OBR), where we could install, update, and uninstall bundles as needed. Then, after having updated the auction bundle, we could select an OSGi framework instance and tell it to move forward and take the new versions of the bundles from the OBR. If there are any problems, we could always tell it to revert to the previous versions.

Let's take a step back and revise. We have several OSGi bundles that have been uploaded to the cloud's OBR. In addition, we've provisioned OSGi framework instances to computing nodes, as shown in figure 10.12. For example, we have two OSGi framework instances in the client tier; therefore, we could assign each to a different computing node. Next, we have the remaining eight OSGi instances, which we could likewise assign to a different computing node. Finally, we could install the web application in the OSGi instances of the client tier and the auction application bundles in the OSGi instances of the business tier. And, of course, we would need to make sure that the web application bundle could invoke remote services residing in the bundles of the business tier. In this environment, we can easily manage updates to our system by leveraging the OSGi framework. But there's still one problem to consider: we're having to explicitly provision the OSGi framework instances and target the bundles to specific framework instances.

A better scenario would be the following: we upload the bundles to the OBR in the cloud, and each bundle tells the cloud its requirements, so that the infrastructure can

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0083

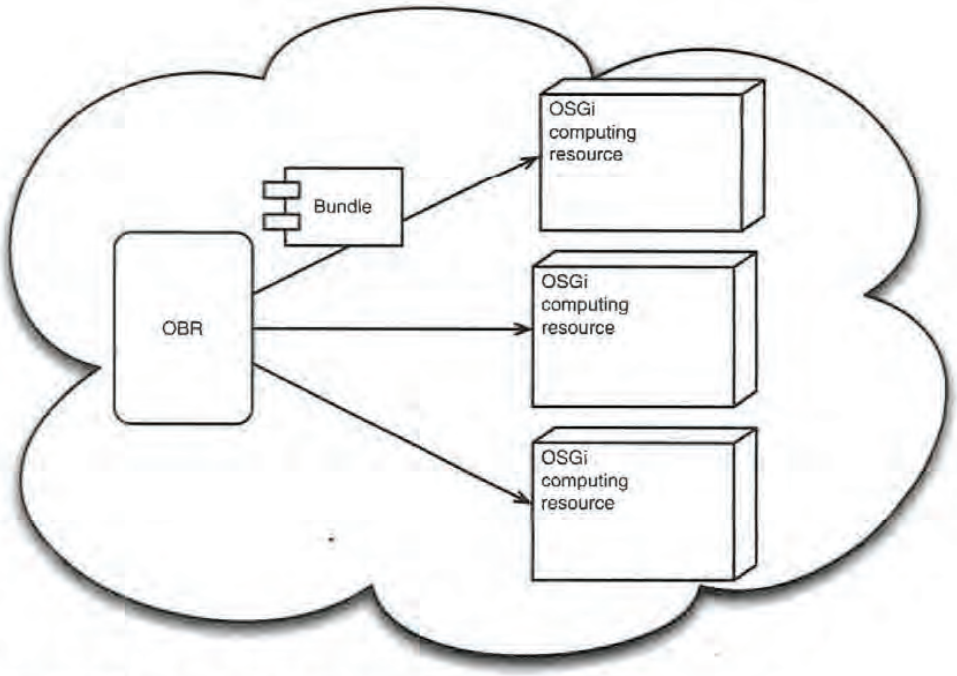


Figure 10.12 OBR and OSGi framework as a platform for cloud computing

dynamically provision the system. For example, the web application bundle could state that it depends on a web server and a cloud resource that has fast public access to the internet. Likewise, the auction bundles could state that they depend on a cloud resource where they get at least 80% CPU utilization.

The cloud infrastructure would consider all these requirements and assign the bundles to the correct OSGi framework instances that it provisions to its computing nodes. For example, the cloud infrastructure would spawn a new computing node that has internet access and install the web application bundle into it.

REQUIREMENTS AND CAPABILITIES

Does it seem far-fetched? Perhaps, but we already have most of the pieces. Dependencies between bundles can already be specified in OSGi using the regular `import-package` mechanism. Furthermore, starting in version 4.3 of the OSGi Service Platform, there's a generic mechanism for the specification of general requirements and capabilities. You can think of the `import-package` and `export-package` headers respectively as a particular case of a requirement and a capability specification.

For example, consider a bundle that specifies the following OSGi 4.3 `Require-Capability` MANIFEST header entry:

```
Require-Capability:
    com.cloudprovider;    filter:="(&(web-access)(cpu<90))"
```

This header is placed in the namespace of `com.cloudprovider`, which means that its interpretation is intended for some particular cloud provider. It states that this bundle

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0084

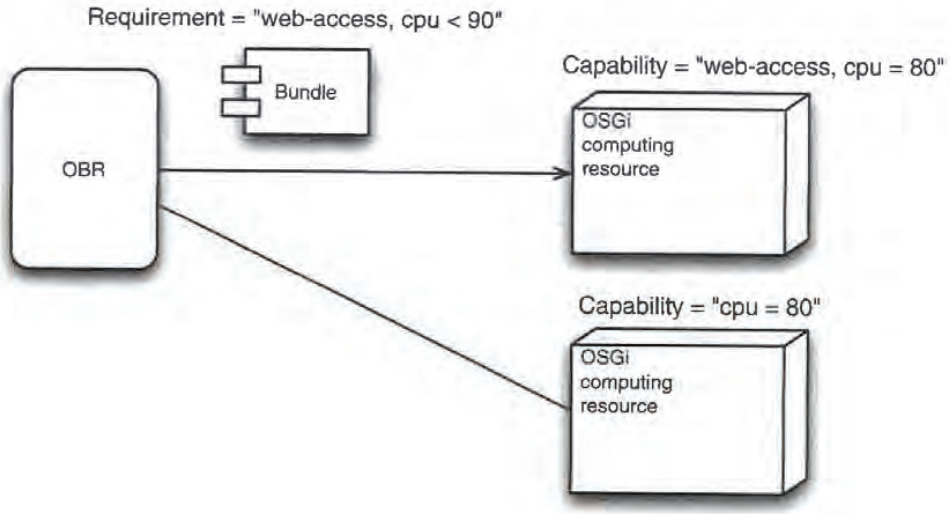


Figure 10.13 The cloud infrastructure matches a bundle's requirements with a resource node that's able to fulfill them.

requires the attribute `web-access` to be true and that the attribute `cpu`, which is a numeric value, must be less than 90 (percent).

On the opposite side, the cloud provider has to find an OSGi framework instance whose System bundle has been launched with the following property:

```
org.osgi.framework.system.capabilities.extra=
        "com.cloudprovider; web-access:Boolean=true; cpu:Long=80"
```

Essentially, the framework instance is telling us that it's capable of providing web access and it will monitor the CPU to make sure that it's never more than 80% busy. This is shown in figure 10.13.

There's still much to be collaborated before cloud computing is considered a mature technology. But the OSGi platform has three intrinsic qualities that make it ideal for supporting elasticity in the cloud:

- Application behavior dynamism provided by the OSGi framework API
- Transport abstraction through OSGi Remote Services
- Dependency management using the OSGi requirement/capability framework

In other words, cloud providers could expose the OSGi service platform as a service to their clients, so that they can manage their applications. In this environment, an OSGi bundle becomes the cloud's deployment unit, and OSGi the ideal Platform as a Service.

10.5 Summary

The OSGi Remote Service specification allows a consuming bundle to invoke a service that's provided by a bundle in a remote OSGi framework instance. You register a service using the service properties `service.exported.interfaces` and `service.exported.configs` to mark it as remote.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0085

Distribution providers are responsible for transparently discovering and handling the remote services for the consuming bundles, and they support different transport protocols, as dictated by the `service.exported.configs` property. A remote service may be exported through different endpoints. A consuming bundle may optionally select a particular endpoint by filtering on the service property `service.imported.configs`.

It's not uncommon for the consumer and the provider of the remote services to negotiate policies, such as if encryption is necessary or based on the QoS of the transport itself. For this purpose, the provider can specify its requirements in the `service.exported.intents` property. The distribution providers must be able to fulfill these requirements, which are seen as capabilities in the consuming side in the service property `service.intents`.

Cloud computing is all about being able to efficiently meet the varying demand of customers by dynamically provisioning resources from a resource pool. The OSGi platform is an ideal Platform as a Service, because it has native support for the dynamic behavior of applications, remote services, and a generic requirements/capability framework for dependency resolution.

In the next chapter, we'll look into the crucial issue of how to manage this complex environment. Managing the OSGi framework and its applications is the subject of the next chapter.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0086

JAVA

LC CIP
0 028 873 659 5

OSGi IN DEPTH

Alexandre de Castro Alves

Free eBook
SEE INSERT

OSGi is a mature framework for developing modular Java applications. Because of its unique architecture, you can modify, add, remove, start, and stop parts of an application without taking down the whole system. You get a lot of benefit by mastering the basics, but OSGi really pays off when you dig in a little deeper.

OSGi in Depth presents practical techniques for implementing OSGi, including enterprise services such as management, configuration, event handling, and software component models. You'll learn to custom-tailor the OSGi platform, which is itself modular, and discover how to pick and choose services to create domain-specific frameworks for your business. Also, this book shows how you can use OSGi with existing JEE services, such as JNDI and JTA.

What's Inside

- Deep dives into modularization, implementation decoupling, and class-loading
- Practical techniques for using JEE services
- Customizing OSGi for specific business domains

Written for Java developers who already know the basics, *OSGi in Depth* picks up where *OSGi in Action* leaves off.

Alexandre Alves is the architect for Oracle CEP, coauthor of the WS-BPEL 2.0 specification, and a member of the steering committee of EP-TS.

For access to the book's forum and a free eBook for owners of this book, go to manning.com/OSGiinDepth

"Gives you a deep understanding of OSGi."
—From the Foreword by David Bosschaert, Red Hat

"Crucial if you're developing OSGi-based systems."
—Steve Gutz, IBM

"A thorough book on an increasingly important topic."
—Rick Wagner, Red Hat

"Alex has loads of experience to back up his words."
—Mike Keith, Oracle

"A jumpstart resource."
—Benjamin Muschko
Webs Inc.

ISBN 13: 978-1-935182-17-7
ISBN 10: 1-935182-17-X
57500
9 781935 182177



MANNING $75.00 / Can $78.99 [INCLUDING eBOOK]

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0087

Exhibit 003

QA 76
.73
.J38
0827
2011
Copy 1



Creating modular applications in Java

OSGi IN ACTION

Richard S. Hall
Karl Pauls
Stuart McCulloch
David Savage

FOREWORD BY PETER KRIENS

MANNING

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0089

OSGi in Action

Creating Modular Applications in Java

RICHARD S. HALL
KARL PAULS
STUART McCULLOCH
DAVID SAVAGE



MANNING
Greenwich
(74° w. long.)

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0090

For online information and ordering of this and other Manning books, please visit www.manning.com. The publisher offers discounts on this book when ordered in quantity. For more information, please contact

Special Sales Department
Manning Publications Co.
180 Broad Street, Suite 1323
Stamford, CT 06901
Email: orders@manning.com

©2011 by Manning Publications Co. All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by means electronic, mechanical, photocopying, or otherwise, without prior written permission of the publisher.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in the book, and Manning Publications was aware of a trademark claim, the designations have been printed in initial caps or all caps.

♾ Recognizing the importance of preserving what has been written, it is Manning's policy to have the books we publish printed on acid-free paper, and we exert our best efforts to that end. Recognizing also our responsibility to conserve the resources of our planet, Manning books are printed on paper that is at least 15 percent recycled and processed without the use of elemental chlorine.



Manning Publications Co.
180 Broad Street, Suite 1323
Stamford, CT 06901

Development editor: Cynthia Kane
Copyeditor: Tiffany Taylor
Typesetter: Gordan Salinovic
Illustrator: Martin Murtonen
Cover designer: Marija Tudor

ISBN 9781933988917
Printed in the United States of America
1 2 3 4 5 6 7 8 9 10 – MAL – 16 15 14 13 12 11



about this book

The OSGi specifications are well written and elaborate, so if you need to know details about OSGi technology, the specifications are the place to look. If you do, you'll discover that they were written for someone who is going to implement the specifications, not use them. This book started out as an attempt to remedy this situation by creating a user-oriented companion guide for the specifications. Our goal wasn't to create an OSGi cookbook but to thoroughly describe the important aspects of OSGi and show how to use them. Our main idea was to more simply explain the OSGi specifications by ignoring the implementation details and including additional usage information.

To that end, we've tried to limit ourselves to discussing the most common concepts, features, and mechanisms needed to work with OSGi technology throughout the book. That doesn't mean we were able to avoid all the esoteric details. As you'll find when you begin working with OSGi, it enforces a new level of strictness when it comes to modularity, which will likely break some of your old practices. In the end, you need to understand what's going on under the covers in some places to be able to effectively debug and diagnose the situations in which you find yourself.

As our writing progressed, the book chapters began to separate naturally into three parts:

1. Explaining the core OSGi specification
2. Describing how to work with the specification in practice
3. Introducing advanced OSGi-related topics

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0092

But these techniques are inherently brittle and error prone because they aren't enforceable via any compile-time or execution-time checks. The end result has detrimental impacts on multiple stages of an application's lifecycle:

- *Development*—You're unable to clearly and explicitly partition development into independent pieces.
- *Deployment*—You're unable to easily analyze, understand, and resolve requirements imposed by the independently developed pieces composing a complete system.
- *Execution*—You're unable to manage and evolve the constituent pieces of a running system, nor minimize the impact of doing so.

It's possible to manage these issues in Java, and lots of projects do so using the custom techniques mentioned earlier, but it's much more difficult than it should be. We're tying ourselves in knots to work around the lack of a fundamental feature. If Java had explicit support for modularity, then you'd be freed from such issues and could concentrate on what you really want to do, which is developing the functionality of your application.

Welcome to the OSGi Service Platform. The OSGi Service Platform is an industry standard defined by the OSGi Alliance to specifically address the lack of support for modularity in the Java platform. As a continuation of its modularity support, it introduces a service-oriented programming model, referred to by some as *SOA in a VM*, to help you clearly separate interface from implementation. This chapter will give you an overview of the OSGi Service Platform and how it helps you create modular and manageable applications using an interface-based development model.

When we've finished this chapter, you'll understand what role OSGi technology plays among the arsenal of Java technologies and why Java and/or other Java-related technologies don't address the specific features provided by OSGi technology.

1.1 The what and why of OSGi

The $64,000 question is, "What is OSGi?" The simplest answer to this question is that it's a modularity layer for the Java platform. Of course, the next question that may spring to mind is, "What do you mean by *modularity*?" Here we use *modularity* more or less in the traditional computer-science sense, where the code of your software application is divided into logical parts representing separate concerns, as shown in figure 1.1. If your software is modular, you can simplify development and

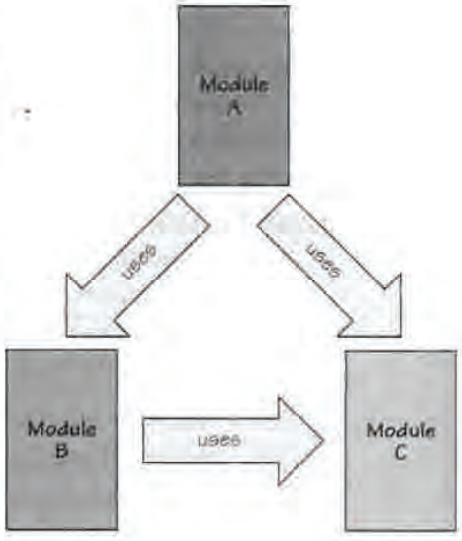


Figure 1.1 *Modularity* refers to the logical decomposition of a large system into smaller collaborating pieces.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0093

improve maintainability by enforcing the logical module boundaries; we'll discuss more modularity details in chapter 2.

The notion of modularity isn't new. The concept became fashionable back in the 1970s. OSGi technology is cropping up all over the place—for example, as the runtime for the Eclipse IDE and the GlassFish application server. Why is it gaining popularity now? To better understand why OSGi is an increasingly important Java technology, it's worthwhile to understand some of Java's limitations with respect to creating modular applications. When you understand that, then you can see why OSGi technology is important and how it can help.

1.1.1 Java's modularity limitations

Java provides some aspects of modularity in the form of object orientation, but it was never intended to support coarse-grained modular programming. Although it's not fair to criticize Java for something it wasn't intended to address, the success of Java has resulted in difficulty for developers who ultimately have to deal with their need for better modularity support.

Java is promoted as a platform for building all sorts of applications for domains ranging from mobile phone to enterprise applications. Most of these endeavors require, or could at least benefit from, broader support for modularity. Let's look at some of Java's modularity limitations.

LOW-LEVEL CODE VISIBILITY CONTROL

Although Java provides a fair complement of access modifiers to control visibility (such as `public`, `protected`, `private`, and package private), these tend to address low-level object-oriented encapsulation and not logical system partitioning. Java has the notion of a *package*, which is typically used for partitioning code. For code to be visible from one Java package to another, the code must be declared `public` (or `protected` if using inheritance). Sometimes, the logical structure of your application calls for specific code to belong in different packages; but this means any dependencies among the packages must be exposed as `public`, which makes them accessible to everyone else, too. Often, this can expose implementation details, which makes future evolution more difficult because users may end up with dependencies on your nonpublic API.

To illustrate, let's consider a trivial "Hello, world!" application that provides a public interface in one package, a private implementation in another, and a main class in yet another.

Listing 1.1 Example of the limitations of Java's object-orientated encapsulation

Greeting.java

```
package org.foo.hello;

public interface Greeting {
  void sayHello();
}
```

❶ Simple interface

GreetingImpl.java

```
package org.foo.hello.impl;

import org.foo.hello.Greeting;
```

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0094

In addition to merging the exported and imported packages and required bundles, the bundle class paths are also merged. This impacts the overall class search order for the bundle, like this:

1. Requests for classes in `java.` packages are delegated to the parent class loader; searching stops (section 2.5.4).
2. Requests for classes in an imported package are delegated to the exporting bundle; searching stops (section 2.5.4).
3. Requests for classes in a package from a required bundle are delegated to the exporting bundle; searching continues with a failure (section 5.3.1).
4. The host bundle class path is searched for the class; searching continues with a failure (section 2.5.4).
5. Fragment bundle class paths are searched for the class in the order in which the fragments were installed. Searching stops if found but continues through all fragment class paths and then to the next step with a failure.
6. If the package in question isn't exported or required, requests matching any dynamically import package are delegated to an exporting bundle if one is found. Searching stops with either a success or a failure (section 5.2.2).

This is the complete bundle class search order, so you may want to mark this page for future reference! This search order makes it clear how fragments support split packages, because the host and all fragment class paths are searched until the class is found.

Fragments and the `Bundle-ClassPath`

The bundle class path search order seems fairly linear, but fragments do introduce one exception. When specifying a bundle class path, you can specify embedded JAR files, such as

```
Bundle-ClassPath: .,specialized.jar,default.jar
```

Typically, you'd expect both of these JAR files to be contained in the JAR file of the bundle declaring them, but this need not be the case. If fragments aren't involved, the framework ignores a non-existent JAR file on the bundle class path. But if the bundle has fragments attached to it, the framework searches the fragments for the specified JAR files if they don't exist in the host bundle.

In the example, imagine that the host contains default.jar but doesn't contain specialized.jar, which is contained in an attached fragment. The effect this has on the class search order is that the specified fragment content is searched before some of the host bundle content.

Sometimes this is useful if you want to provide default functionality in the host bundle but be able to override it on platforms where you have specialized classes (for example, using native code). You can also use this approach to provide a means for issuing patches to bundles after the fact, but in general it's better to update the whole bundle.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0095

8.3 Tracking down memory leaks

Memory leaks occur in OSGi applications as in any other Java application. All you need is something like a rogue thread or static field hanging on to one end of a spaghetti ball of references to stop the garbage collector from reclaiming the objects. In a desktop Java application, you may not notice any memory leaks because you don't leave the application running for a long time. As soon as you restart the JVM, your old application with its ever-growing heap of objects is gone, and you get a brand-new empty heap to fill.

Server-side OSGi applications, on the other hand, can have longer lifetimes; an uptime of many months isn't unreasonable. One of the strengths of OSGi is that you're able to install, update, and uninstall bundles without having to restart the JVM. Although this is great for maximizing uptime, it means you have to be careful not to introduce memory leaks in your bundles. You can't always rely on the process being occasionally restarted. Furthermore, updating a bundle introduces a new class loader to hold the updated classes. If there's anything holding on to objects or classes from the old class loader, it won't be reclaimed, and your process will use more and more class loaders each time the bundle is updated or reinstalled.

Introducing the PermGen heap

Class-loader leaks can be more problematic than simple object leaks because some Java runtimes, like Sun's HotSpot JVM, place classes in a separate heap space called the Permanent Generation, or PermGen for short. This class heap is much smaller than the main object heap, and its capacity is controlled by a different GC setting: `-XX:MaxPermSize`. If every bundle update adds hundreds of new class revisions without unloading anything, you'll probably exhaust the PermGen before you run out of object heap space.

Any leak is a cause for concern, but depending on your requirements, not all leaks warrant investigation. You may not even notice certain leaks if they add only a few bytes to the heap every now and then. What's the best way to find leaks in an OSGi application?

8.3.1 Analyzing OSGi heap dumps

As with debugging, you can continue to use your existing heap analysis skills to examine OSGi applications. Sure, there are more class loaders than in a normal Java application; but standard Java EE containers also contain multiple class loaders, and that doesn't stop developers from finding memory leaks inside web applications.

Let's see what an OSGi application heap dump looks like. The leaky application is under chapter08/memory-leaks in the online examples. It consists of a single bundle that creates and accesses a `ThreadLocal` variable every time the bundle starts and fails to remove it when the bundle stops. Here's the bundle activator.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0096

To be able to show all of what deployment packages have to offer, let's assume you want to provide a core version of the program containing the drawing frame and the shape API bundles. This way, you're able to deploy the actual shape implementations separately via an extension pack. The extension pack contains the square, circle, and triangle bundles. Let's go with this approach and explore the different ways you can use deployment packages to make it work.

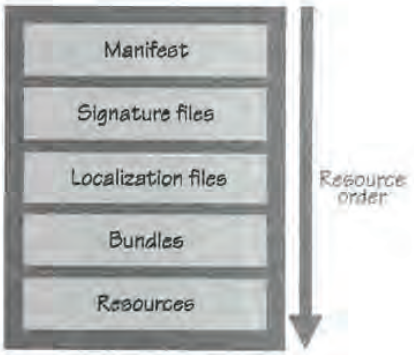


Figure 10.4 Structure of a deployment package JAR file

The general structure of a deployment package is shown in figure 10.4. This ordering is carefully designed to allow deployment packages to be streamed in such a way that the contents can be processed without needing to download the entire JAR file.

The deployment package design has a few other desirable characteristics. First, the deployment package puts metadata in its manifest, similar to bundles, which allows you to turn it into a named and versioned set of resources. Second, by taking advantage of the fact that JAR files can be signed, you can use signed JAR files to make your deployment packages tamperproof.

For this example, you can do either of the following (see figure 10.5):

- Create a deployment package for the core bundles and one package for all shape bundles.
- Create a deployment package for the core bundles and individual deployment packages for each shape bundle.

The difference is obviously that in the first case, you'll deploy either all shapes or none; and in the second case, you can extend the core bundle piecemeal. The important point to understand, though, is that you can't use both approaches at the same time: you must choose one.

In terms of the example, you need to make a decision. In this case, you'll go with the first approach and create a single deployment bundle for all shapes. But because deployment packages can be updated, you can gain some flexibility by starting with only one shape in the deployment package and then adding another one in an updated version and

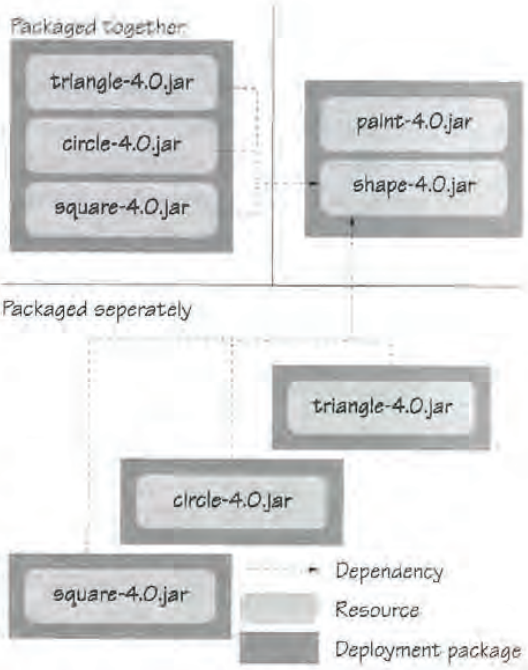


Figure 10.5 Paint program packaging alternatives

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0097

Deployment packages are greedy

These two different packaging strategies can't be used simultaneously. The specification only allows resources to belong to a single resource package. Using both approaches at the same time or changing your approach after the fact would move ownership of the bundle resources to another deployment package and thus violate the specification.

A deployment package is defined as a set of resources that must be managed as a unit. The resources in a deployment package are assumed to be tightly coupled, such as a bundle and its configuration data. As a consequence, a resource can belong to only one deployment package; otherwise, for example, you could run into situations where you had two different, conflicting configurations for the same bundle.

another for the third or other combinations. When you create an update that adds or removes resources from a previous version, you don't even have to package the resources inside the update; instead, you can use *fix packages.*

FIX PACKAGE A deployment package that minimizes download time by excluding resources that aren't required to upgrade or downgrade a deployment package. It can only be installed if a previous version of that deployment package is already installed. A fix package contains only the changed and new resources. A fix package (called the *source*) therefore must specify the range of versions that the existing deployment package (called the *target*) must have installed. You'll see this shortly when we walk through the example.

Let's assume that you want to be able to add new shapes to the application when they become available. In this scenario, it makes sense to start with a core deployment package and create fix packages, adding new shapes as they become available.

Now that you've figured out your packaging approach, how do you proceed? You need to create a manifest for the target that contains the paint frame and shape API bundles; you'll use this to provision the paint program core. Then you need to create the manifest of the fix package that you'll use to add the three shape bundles to the core. When you have your manifests, you need to create two JAR files with the corresponding manifests and your bundles, you can optionally sign them, and you're good to go. Here's the manifest of the core deployment package:

```
Manifest-Version: 1.0
DeploymentPackage-SymbolicName: org.foo.paint
DeploymentPackage-Version: 1.0.0

Name: paint-4.0.jar
Bundle-SymbolicName: org.foo.paint
Bundle-Version: 4.0.0

Name: shape-4.0.jar
Bundle-SymbolicName: org.foo.shape
Bundle-Version: 4.0.0
```

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0098

You first specify the deployment package's symbolic name and version. Next, you specify the list of resources contained in the JAR file. You specify the name of a resource, its symbolic name, and its version; you must do this for each resource. For this example, you only have bundle resources. To finish, you need to use the `jar` tool to create the JAR file with the appropriate content, and you're finished with your first deployment package.

Signing deployment packages

In this example, you don't sign your deployment package, nor is it required for you to do so. If you want to create a signed deployment package, you use the `jarsigner` tool from the standard Java SDK. The signing process is no different than signing a normal JAR file; it results in the signatures being placed in the deployment package JAR file in the META-INF directory and after the MANIFEST.MF file. Additionally, each entry section in the manifest contains a digest entry.

Now you need to create the manifest for your fix package containing the shape bundles. This manifest is as follows:

```
Manifest-Version: 1.0
DeploymentPackage-Symbolicname: org.foo.paint
DeploymentPackage-Version: 2.0
DeploymentPackage-FixPack: [1,2)

Name: paint-4.0.jar
Bundle-SymbolicName: org.foo.paint
Bundle-Version: 4.0.0
DeploymentPackage-Missing: true

Name: shape-4.0.jar
Bundle-SymbolicName: org.foo.shape
Bundle-Version: 4.0.0
DeploymentPackage-Missing: true

Name: triangle-4.0.jar
Bundle-SymbolicName: org.foo.shape.triangle
Bundle-Version: 4.0.0

Name: circle-4.0.jar
Bundle-SymbolicName: org.foo.shape.circle
Bundle-Version: 4.0.0

Name: square-4.0.jar
Bundle-SymbolicName: org.foo.shape.square
Bundle-Version: 4.0.0
```

Because the fix package is an update to your core package, the symbolic name stays the same, but the version is upgraded to `2.0.0`. The `DeploymentPackage-FixPack` header indicates that this is a fix package; you use version-range syntax to indicate that the fix package can be applied to any previously installed version of the deployment package from `1.0.0` inclusive to `2.0.0` exclusive. This version-numbering scheme expresses the assumption that only major version-number changes indicate added

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0099

bundles. You don't need to package the bundles already present in the core package, but you still need to mention them in the manifest. You use the `DeploymentPackage-Missing` header to do this. Then you specify the shape bundles in the same fashion as before. To use the deployment packages, you need to make each available via a URL.

> **NOTE** If you make deployment packages available via a protocol that supports MIME types, the standard MIME type for deployment packages is `application/vnd.osgi.dp`.

Next, you can use the provided `DeploymentAdmin` service in your management agent to install, update, and uninstall deployment packages.

MANAGING DEPLOYMENT PACKAGES

To demonstrate how a management agent can use Deployment Admin, you'll again return to the shell and create a new dpa shell command to list, install, and uninstall deployment packages. This command will use the `DeploymentAdmin` service, which is represented by the following interface:

```
public interface DeploymentAdmin {
  DeploymentPackage installDeploymentPackage(InputStream in)
    throws DeploymentException;
  DeploymentPackage[] listDeploymentPackages();
  DeploymentPackage getDeploymentPackage(String symbName);
  DeploymentPackage getDeploymentPackage(Bundle bundle);
  boolean cancel();
}
```

The following listing shows the implementation of the command.

Listing 10.4 Deployment Admin shell command example

```
public class DeploymentPackageCommand extends BasicCommand {

  public void exec(String args, PrintStream out, PrintStream err)
    throws Exception {
    DeploymentAdmin admin = getDeploymentAdmin();

    if (admin == null) {
      out.println("No DeploymentAdmin service found.");
      return;
    }
    if (args != null) {
      if (args.trim().equalsIgnoreCase("list")) {
        for (DeploymentPackage dp : admin.listDeploymentPackages()) {
          out.println(dp.getName() + " " + dp.getVersion());
        }
      } else if (args.trim().startsWith("uninstall ")) {
        DeploymentPackage dp = admin.getDeploymentPackage(
          args.trim().substring("uninstall ".length()));
        if (dp != null) {
          dp.uninstall();
        } else {
          out.println("No such package");
        }
```

❶ Gets installed deployment packages

❷ Uninstalls deployment package

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0100

14 Securing your applications

This chapter covers

- Providing an overview of the Java security architecture
- Using OSGi Conditional Permission Admin to manage permissions
- Signing bundles and granting permissions based on bundle signers
- Creating custom permission conditions for advanced use cases

OSGi allows you to create loosely coupled, extensible applications. In such applications, bundles can come and go at any time, and it's easy to allow third parties to extend your application in a well-defined way. But as with most things in life, there's a downside to this flexibility: you open yourself (or your users) to security vulnerabilities because third-party bundles can't be completely trusted.

Luckily, the Java platform has built-in support for secure sandboxes, and the OSGi framework is designed to take advantage of them. Unfortunately, secure

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0101

sandboxes and their restrictions are difficult to get right and often hard to manage. This is especially true in an environment as dynamic as the OSGi framework. To help with this situation, OSGi defines an extensive and powerful security model to ease security management.

In this chapter, we'll familiarize you with the Java security model and how OSGi uses it to securely deploy and manage applications. You'll learn how to create secure applications and well-behaved bundles for security-enabled OSGi frameworks. Before we start with that, let's look at some general issues you'll need to consider when trying to secure your applications.

14.1 To secure or not to secure

Modern applications and software solutions increasingly center around loosely coupled, extensible architectures. Component and service orientation are applied to almost all areas of application development including distributed systems, ubiquitous computing, embedded systems, and client-side applications. One of the main drawbacks of loosely coupled, extensible applications is the potential security issues around executing untrusted code. There are two common reasons for running untrusted software:

- *Permission management is often extremely complicated.* Often, users are left to make security policy decisions, and they're typically unable to assess the impact of granting a given permission. Further, because the user is typically using an application to perform some task, security is largely viewed as an obstacle because it doesn't contribute to the task at hand.
- *It's inherently tricky to establish meaningful identity of third-party software providers.* It's often necessary to differentiate between providers or types of providers to properly grant or deny permissions. Often the origin of the software artifact (where it came from) is used for this purpose, but techniques like digital signatures are also needed to ensure that the software hasn't been tampered with. Digital signatures introduce the complicated process of creating and maintaining certificates and trust between certificates, which can be onerous for both users and developers.

This raises perhaps the biggest issue with securing code: it adds another burden to development. Even if you don't plan to run with security enabled, your code has to be aware of security if you want it to be possible for other people to use the code when security is enabled. Then, to make matters worse, if you decide to enable security, the fine-grained security checks impose an execution-time performance penalty.

Despite these issues, security isn't something that can or should be ignored, because plenty of people are willing to take advantage of software vulnerabilities. What do you need to do? Providing meaningful security management involves three key aspects:

- Establishing identity
- Establishing policies to manage permissions
- Performing permission checks and privilege management

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0102

As we mentioned previously, identity can be established by the location or origin of the software artifact or by cryptographic measures using digital certificates. Especially for the latter approach, the software provider generally needs to make the code available in such a way that you can establish the needed credentials. When you have identity established, you need to define the permissions that code should have. For OSGi, this is the responsibility of whoever is managing the framework, which can be a gateway operator, a server administrator, or even an end user. As a consequence, permission management should be kept as simple as possible. Last but not least, security must be built into the code itself. You have to think about internal security checks to prevent external code from performing undesired operations and also how to limit privileges so external code can perform potentially sensitive operations in a safe way.

Assuming you're able to develop your code with all the security checks in the right place, define a reasonably policy to manage permissions, and sign it using a trusted certificate for establishing identity, is all the work worth it?

Clearly, it depends on your specific situation. In some cases, it's not within the scope of your application. Either the performance impact is too great or the development costs are too high. Often, these issues serve as the determining factor for creating security-enabled applications. This is compounded by the fact that if code isn't *designed* to be usable in security-enabled environments, it's unlikely to happen by accident. This results in a catch-22 type of situation, where the difficulty associated with creating secure code results in security being ignored, which makes it next to impossible to use such code with security enabled, thus further raising the barriers for deciding to develop with security in mind in the first place.

All hope isn't lost. In the remainder of this chapter, we'll show you that taking advantage of the security capabilities of the OSGi framework needn't be too difficult. In the next section, we'll start by taking a high-level view of Java and OSGi security.

14.2 Security: just do it

So you want to secure your OSGi-based application. Great—but where do you start? Let's begin at the beginning and look at the Java security architecture and its permission model, on which the OSGi security model is based.

14.2.1 Java and OSGi security

It's important to understand the Java security architecture; but to keep this chapter tightly scoped, we'll introduce only the parts needed to understand the remainder of the chapter. Welcome to Java security boot camp!

The Java security architecture is fundamentally about the assignment of `java.security.Permission` subclass objects to code. Specific permissions allow access to specific, sensitive operations, such as file system or network socket access. How do you grant `Permission` objects to code? The Java security architecture is based on the two fundamental concepts of domain- and role-based security:

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0103

- Domain-based security revolves around granting code permissions based on its origins (also referred to as its *code base*).
- Role-based security revolves around authenticating users or processes and granting them permissions based on who they are.

The OSGi framework security model relies on Java's domain-based approach; the role-based approach is possible, but only as a layer on top. In standard Java, role-based security is provided by the Java Authentication and Authorization Service (JAAS) framework, but OSGi also provides its own API in the form of the User Admin Service. We won't deal with role-based security in this chapter; for more information on the User Admin Service, refer to the OSGi compendium specification. Now, let's delve a little deeper into domain-based security.

PERMISSIONS

The Java permission model is fairly simple. The `Permission` class is a base class from which more specific permissions can be derived via subclassing. You grant `Permission` objects to code to give it the ability to perform sensitive operations. Additionally, the `Permission` class has a method called `implies()` that accepts another `Permission`. This method checks to see if the supplied permission is implied by the target permission (similar to being a subset). Thus, `Permission` objects are used to both grant and check permissions.

PROTECTION DOMAINS

You grant permissions to code, but how are they associated with it? For domain-based security, Java uses a special concept called a *protection domain*, which is represented by the `java.security.ProtectionDomain` class, to encapsulate the security characteristics of a domain. Permissions are granted to protection domains, and all classes belong to a single protection domain. Sound complicated? Actually, in OSGi it's pretty simple, because a domain is mapped one-to-one with a bundle; you can think of it as a bundle protection domain. All classes originating from a given bundle are members of the same bundle protection domain.

> **BUNDLE PROTECTION DOMAIN** Maintains a set of permissions granted to a given bundle. All classes loaded from a bundle are associated with the bundle's protection domain, thus granting them the permissions granted to the bundle.

To understand how protection domains enable permission checking, consider code that performs a sensitive operation, such as creating a file. The code in the JRE for file access performs security checks internally to make sure the invoking code has permission to perform the operation. Internally, the code associated with performing file system operations triggers a specific permission check by using the security-checking methods of `SecurityManager` or `AccessController`. When triggered, the JVM collects the `ProtectionDomains` of all classes on the call stack leading to the invocation of the sensitive operation. It checks that each protection domain on the call stack has at least one permission implying (granting) the specific permission being checked by the method. Figure 14.1 shows how this looks in practice.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0104

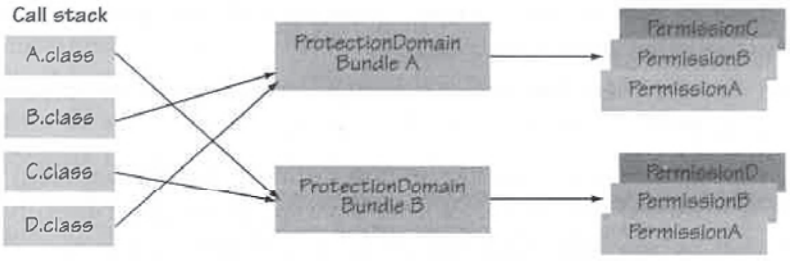


Figure 14.1 The JVM checks permissions by collecting all protection domains associated with classes on the call stack and seeing if each involved protection domain has the specified permission granted to it.

In this case, assume that class `D` performs a sensitive operation that triggers an internal permission check using the `AccessController.checkPermission(Permission p)` method. This checks whether at the point of the permission check, all protection domains on the call stack have permission `p`. Looking at figure 14.1, the JVM performs a stack walk from the class performing the security check and determines that classes `A`, `B`, `C`, and `D` are involved. Subsequently, it determines that classes `A` and `C` originate from the protection domain of Bundle B, and classes `B` and `D` originate from the protection domain of Bundle A. With this information, the JVM checks whether all protection domains have some permission implying the checked permission. If not, then a security exception is thrown.

This provides a good foundation for understanding the Java security architecture, but there's one final piece to the puzzle: privileged calls.

PRIVILEGED CALLS

You now know that checking a specific permission triggers a complete stack walk to collect all involved protection domains and verify that they all imply that permission. This is useful, but it's too restrictive by itself. For example, assume you have a service with a method for appending a message to a log file. Because disk operations trigger file system–related permission checks, all code on the call stack must have permission to write to the file. This may be fine if only trusted code is involved; but in an extensible and collaborative environment like OSGi, you generally want to allow arbitrary bundles to share code and services, so it's likely that some code on the call stack won't be trusted.

In such cases, if a permission check always walks up the entire call stack, you either have to disallow all nontrusted code or grant code-sensitive permissions to untrusted code. Neither choice is palatable, which is why Java supports privileged calls. A *privileged call* is a mechanism to short-circuit the stack walk when performing a permission check. In practice, this allows trusted code to perform sensitive operations on behalf of code with insufficient permissions.

You achieve this by using the `AccessController.doPrivileged()` method, which takes a `PrivilegedAction` instance (it has a `run()` method similar to a `Runnable`). When the `doPrivileged()` method is invoked, it invokes the `run()` method of the passed-in `PrivilegedAction`. Any subsequent permission checks triggered by the `PrivilegedAction` stop walking the call stack at the last privileged call. Thus, only the protection domains from the privileged action onward are considered by subsequent permission checks.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0105

Returning to the example of a service for appending a message to a log file, you trust the bundle containing the service implementation, but you don't want to give direct file system access to anyone else. To do this, your service must encapsulate its file system operations inside a `PrivilegedAction` and use `doPrivileged()` like this:

```
public void append(String msg) {
  SecurityManager sm = System.getSecurityManager();
  if (sm != null) {
    AccessController.doPrivileged(new PrivilegedAction() {
      public Object run() {
        doFileAppend();
      }
    });
  }
  else {
    doFileAppend();
  }
}
```

Any triggered permission checks stop walking the call stack at the `run()` method, which means nontrusted code further up the stack won't have its protection domain checked for the triggered permissions. Pushing this example further, you may decide to limit which code can call the `append()` method. To so this, you can create your own `Permission` subclass, which you can grant to code. For the append method, if you create an `AppendPermission`, it can check the permission before performing the privileged call:

```
public void append(String msg) {
  SecurityManager sm = System.getSecurityManager();
  if (sm != null) {
    sm.checkPermission(new AppendPermission());
    AccessController.doPrivileged(new PrivilegedAction() {
      public Object run() { doFileAppend(); }});
  } else {
    doFileAppend();
  }
}
```

Here your service asks the `SecurityManager` to check whether the code on the call stack has been granted the custom `AppendPermission`. If so, it can continue to perform the file-append operation; otherwise, a security exception is thrown.

> **WARNING** You may have noticed that you check whether the security manager is `null` before performing security checks. You do it this way because you want to perform security checks only if security is enabled, to avoid performance penalties when it's not enabled.

That pretty much sums up the important pieces of the Java security architecture. These mechanisms provide for flexible, fine-grained security management. A potential downside is that managing all these permissions can be complex. Luckily, the OSGi specification lessens some of this complexity by defining services to help you perform

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0106

permission management. We'll look at these services shortly; first, let's examine OSGi-specific permissions defined by the OSGi specification.

14.3 OSGi-specific permissions

Certain methods in the OSGi framework API perform sensitive operations or provide access to sensitive information. To control which code can access these sensitive methods, the OSGi specification defines a few custom permissions, as you learned about in the last section. You can group these permissions by the layers of the OSGi framework, as shown in table 14.1.

Table 14.1 Permissions defined by the OSGi specification

Layer	Permission(s)
Module	`PackagePermission`—Controls which packages a bundle is allowed to import and/or export
	`BundlePermission`—Controls which bundles a bundle is allowed to require
Lifecycle	`AdminPermission`—Controls which bundles are allowed to perform sensitive lifecycle operations
Services	`ServicePermission`—Controls which services a bundle is allowed to publish and/or use

We'll introduce these OSGi permissions in the following subsections, and you'll subsequently use them when we discuss permission management.

Names and actions

Standard Java permissions typically have constructors that accept two parameters: a `name` string and an `actions` string. The meaning of these parameters is determined by the specific permission. For example, `java.io.FilePermission` expects a file path for `name` and a comma-delimited value that may include `READ`, `WRITE`, `EXECUTE`, or `DELETE` for actions. The combination of `name` and `actions` allows you to express everything the permission allows you to control. All of the OSGi-specific permissions follow this pattern, as you'll see.

14.3.1 PackagePermission

`PackagePermission` is a module-layer permission giving you the ability to limit which packages a bundle can import or export. For example, we discussed how methods can use `AccessController.doPrivileged()` to allow code with insufficient privileges to perform sensitive operations. You may not want any arbitrary code using the packages containing these privileged operations. In that case, you can use `PackagePermission` to limit which bundles can import the packages containing the associated classes. Likewise, you can use `PackagePermission` to control which bundles can export a given package, because you may only want trusted bundles providing some packages.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0107

To grant a specific `PackagePermission`, you need to supply the `name` and `actions` parameters for its constructor; these parameters are described in table 14.2.

Table 14.2 `PackagePermission` constructor parameters

Parameter	Description
`String name`	Name of the package or packages to which this permission applies
`String actions`	Comma-delimited list of the actions granted by the permission (`export`, `import`, or `exportonly`)

For convenience, you can use `*` or a trailing `.*` as a wildcard to target several packages with a single permission. For the actions, `import` gives a bundle permission to import the named packages, `export` gives a bundle permission to export and import the package, and `exportonly` does as its name implies. You may wonder why `export` also gives permission to import the named packages. It's to support bundles' ability to import packages they export (that is; substitutable exports), as described in section 5.1.1.

To get an idea of how `PackagePermission` works, let's take a conceptual look at how the framework uses it. Assume you have a bundle with the following imports and exports:

```
Import-Package: org.foo, org.bar
Export-Package: org.bar
```

When the framework resolves this bundle, it checks to see whether the bundle has the following permissions granted to it:

- `PackagePermission.IMPORT` permission for the `org.foo` package
- `PackagePermission.IMPORT` permission for the `org.bar` package
- `PackagePermission.EXPORT` permission for the `org.bar` package

For these checks to succeed, you'd have to grant the necessary permissions, such as

```
new PackagePermission("org.foo", PackagePermission.IMPORT);
new PackagePermission("org.bar", PackagePermission.EXPORT);
```

Notice that you don't need to grant the bundle permission to import `org.bar`, because it's implied by the export action.

That's the basics for `PackagePermission`; let's move on to the next OSGi permission.

14.3.2 BundlePermission

Similar to `PackagePermission`, `BundlePermission` is a module-layer permission for controlling bundle and fragment dependencies. To grant a `BundlePermission`, you need to construct it with the parameters shown in table 14.3.

As with `PackagePermission`, you can use `*` or a trailing `.*` as a wildcard to target several packages with a single permission. To control bundle dependencies, the `provide` action gives a bundle permission to be required by bundles matching the supplied

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0108

Table 14.3 `BundlePermission` constructor parameters

Parameter	Description
`String symbolicName`	Symbolic name of the bundle to which this permission applies
`String actions`	Comma-delimited list of the actions granted by the permission (`provide`, `require`, `host`, or `fragment`)

symbolic name, whereas `require` gives it permission to require matching bundles. Fragment dependencies are controlled by the `host` and `fragment` actions, which give a bundle the ability to be a host bundle for matching fragments or be a fragment for matching hosts, respectively. Another similarity to `PackagePermission` is that the `provide` action implies the `require` action.

Using `BundlePermission` isn't sufficiently different from using `PackagePermission`, so we won't look into it any further. Instead, we'll move on to the next OSGi permission.

14.3.3 AdminPermission

`AdminPermission` is a lifecycle-layer permission to control access to sensitive framework operations and information. The operations and information protected by `AdminPermission` are diverse, which makes it somewhat complex but fairly powerful. Table 14.4 shows the parameters needed to create such a permission.

Table 14.4 `AdminPermission` constructor parameters

Parameter	Description
`String filter`	LDAP filter to specify matching bundles
`String actions`	Comma-delimited list of the actions granted by the permission (`class`, `execute`, `extensionLifecycle`, `lifecycle`, `listener`, `metadata`, `resolve`, `resource`, `startlevel`, or `context`)

When you grant `AdminPermission` to a bundle, that bundle is allowed to perform the specified actions on the bundles matching the filter. The filter uses the same LDAP filter syntax used by the OSGi service registry, but only the following attributes are defined for matching:

- `signer`—Identity information about the bundle provider
- `location`—Bundle location
- `id`—Bundle identifier
- `name`—Bundle symbolic name

We'll give some filter examples shortly. First we'll briefly describe what the granted actions allow on the matching bundles:

- `class`—Load a class
- `execute`—Start, stop, and set the start level

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0109

- `extensionLifecycle`—Manage extension bundles
- `lifecycle`—Install, update, and uninstall
- `listener`—Add/remove a synchronous bundle listener
- `metadata`—Retrieve bundle headers and location
- `resolve`—Resolve and refresh
- `resource`—Get resources
- `startlevel`—Set the start level, and initial the start level
- `context`—Get the bundle context

The special action `*` represents all actions. As you can see, `AdminPermission` gives you fine-grained control over which bundles can do what on which bundles. For example, assume a bundle wants to install another bundle using code like this:

```
context.installBundle("file:bundle.jar").start();
```

This triggers the framework to check whether all code on the call stack has the following:

- `AdminPermission.LIFECYCLE` permission for the installed bundle
- `AdminPermission.EXECUTE` permission for the installed bundle

This is relatively straightforward, although granting the permission can be a little confusing. The thing to remember about `AdminPermission` is that you use it to grant a bundle the right to perform specific operations on other bundles. The `filter` constructor parameter is how you specify the bundles that can be controlled. For a more complicated example, you can grant an `AdminPermission` like this:

```
new AdminPermission("(&(signer=CN=core,O=baz,C=de)(name=org.foo.*)
  (location=file://*)(id>=10))", AdminPermission.LIFECYCLE + "," +
  AdminPermission.EXECUTE);
```

The bundle granted this permission can perform the operations associated with `AdminPermission.LIFECYCLE` and `AdminPermission.EXECUTE` on bundles with a signer matching `CN=core,O=baz,C=de` (more on signing later), a symbolic name starting with `org.foo`, a location starting with `file://`, and a bundle identifier greater than 10. Granted, this is completely contrived, but it illustrates the possibilities.

You've now seen the module- and lifecycle-layer permissions, which means you have one framework layer to go—services.

14.3.4 ServicePermission

`ServicePermission` is a service-layer permission for controlling which services a bundle can provide or use. As with the other permissions, the actual permission granted is controlled by its constructor parameters, as shown in table 14.5.

You can use `*` or a trailing `.*` as a wildcard to target several service interfaces or classes with a single permission. The `get` action grants the ability to use the specified services, whereas the `register` action grants the ability to provide the specified services. For the `get` action, you can also use an LDAP filter for `name`, which matches

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0110

Table 14.5 `ServicePermission` constructor parameters

Parameter	Description
`String name`	Service interface or class name(s)
`String actions`	Comma-delimited list of the actions granted by the permission (`get` or `register`)

against the associated service properties in addition to the same set of bundle properties described for `BundlePermission` (`signer`, `location`, `id`, and `name`).

To get a better understanding of how this permission is used, consider the following snippet of code a bundle can use to find a service and to register a service:

```
context.getServiceReference("org.foo.Service");
context.registerService("org.bar.Service", new Service(), null);
```

Here you find an `org.foo.Service` service and register an `org.bar.Service` service. This triggers the framework to check whether all code on the call stack has the following:

- `ServicePermission.GET` for the `org.foo.Service` interface
- `ServicePermission.REGISTER` for the `org.bar.Service` interface

These permission checks are straightforward. For the associated bundle to perform these tasks, you can grant it these permissions:

```
new ServicePermission("org.foo.*", ServicePermission.GET);
new ServicePermission("org.bar.Service", ServicePermission.REGISTER);
```

In the first permission, you use a wildcard to allow it to access all services in the `org.foo` package. In the second permission, you specifically allow it to register `org.bar.Service` services.

That completes the OSGi-specific permissions you can grant to bundles. Before we move on to discussing permission management, let's briefly discuss file permissions, because they behave slightly differently in an OSGi environment.

14.3.5 Relative file permissions

Although `java.io.FilePermission` isn't defined by the OSGi specification, it's still impacted by how the framework interprets it. In a standard Java environment, a file permission created with a relative path is interpreted as being relative to the directory from which the Java process was started (the current working directory). This isn't the case in an OSGi environment. Instead, it's treated as relative to the root of the private data area of the associated bundle.

Typically, this doesn't have much of an effect, especially because the framework automatically grants bundles permission to read, write, and delete files in their own private area. The main thing this enables is the ability to grant a bundle additional permissions for files in its private data area, such as the execute permission.

Enough of describing permissions. Next, we'll discuss how you manage them with the Conditional Permission Admin Service.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0111

```
    + context.getBundle().getLocation() + "\"]"
    + "(java.security.AllPermission \"*\" \"*\")"
    + "} \"Management Agent Policy\"");
  ConditionalPermissionAdmin cpa =
    getConditionalPermissionAdmin(context);
  ConditionalPermissionUpdate u = cpa.newConditionalPermissionUpdate();
  List infos = u.getConditionalPermissionInfos();
  infos.clear();
  for (String encodedInfo : encodedInfos) {
    infos.add(cpa.newConditionalPermissionInfo(encodedInfo));
  }
  if (!u.commit()) {
    throw new ConcurrentModificationException(
      "Permissions changed during update");
  }
}
```

This method combines most of your knowledge about the `ConditionalPermissionAdmin` service. You get the policy file, which defaults to a file called `security.policy` but can be configured. Next, you read in the encodings of the `ConditionalPermissionInfo` objects contained in the policy file and add an encoding for an `AllPermission` at the beginning of the list for the policy-file reader bundle. The previous step is necessary to make sure the policy-reader bundle has sufficient permission to make future changes to the security policy.

After this, you get the `ConditionalPermissionAdmin` service, create a new `ConditionalPermissionUpdate`, and use it to get the current list of `ConditionalPermissionInfo` objects. You clear the existing policy to make sure you're starting with a clean slate, and then loop through the encoded permissions to decode them and add them to your list of objects. The only thing left to do is commit the update. Because the update may fail if the permissions were changed concurrently, you throw an exception in this case.

To see the full details of this bundle, go to the chapter14/combined-example/org.foo.policy/ directory of the book's companion code. This bundle is generic and can be used in any security-enabled framework to put a policy file into effect. You'll see it in action a little later when we show a complete example with digitally signed bundles and a custom condition. We'll introduce bundle signing next.

14.5 Digitally signed bundles

Defining a security policy by assigning permissions to bundles is a workable approach, but being able to step up a level can simplify things. For example, you may trust a particular provider, so it's nice to be able to assign permissions based on the provider rather than individual bundles. Doing so simplifies defining a security policy, because it raises the level of abstraction. Digitally signed bundles can help you achieve this; specifically, they help you do two things:

- Authenticate the provider of a bundle
- Ensure that bundle content hasn't been modified

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0112

The former provides the ultimate goal, but without the latter, the former would be meaningless. You'll learn about both as we discuss digital signing and certificates. We'll show you how to create certificates and use them to digitally sign your bundles. Then we'll introduce `BundleSignerCondition`, which gives you the ability to grant permissions based on the identity established via certificates. First, let's get some terminology out of the way.

14.5.1 Learning the terminology

The domain of digital cryptography is complex and sophisticated. Providing a complete and detailed description is beyond the scope of the book, so we'll focus on describing just enough to have it make sense. With that in mind, table 14.6 introduces some relevant terms we'll use throughout the remainder of the chapter.

Table 14.6 Digital cryptography terminology

Term	Definition
Digital signing	A mathematical approach for verifying the authenticity of digital data. Specifically, used to verify the identity of the provider and that the data hasn't been modified.
Signature	A unique value calculated when data is digitally signed.
Public key cryptography	A form of digital signing using two mathematically related keys: a public key and a private key. The private key is a guarded secret used to sign data. The public key is shared with others in the form of a certificate, which they can use to verify that a signature was generated with the private key. This allows you to infer the identity of the provider and determine whether someone has tampered with the data.
Certificate	A form of metadata about a public key, binding it to the identity of the private key holder. This binding is achieved by having a well-known (trusted) third party sign the public key/identity pair.
Distinguished name	The identification portion of a certificate; specifically, as defined by the X.509 ITU-IT standard. Identifies the holder of the private key.
Certificate chain	A certificate has a reference to the certificate of its third-party signer, which includes a reference to the certificate of its signer, and so on, until the root. This is a certificate chain. The root of the certificate chain is a self-signed certificate.

A general understanding of these terms should be sufficient. You don't need a complete understanding of digital cryptography to use the technology effectively. We'll start looking at the basics.

14.5.2 Creating certificates and signing bundles

You'll be using digital signing based on public key cryptography, which involves a public key and a private key. The public key is shared with the world in the form of a certificate. The private key is kept secret and used to sign data by performing a computation over it. The resulting value can be verified by performing another calculation over the data using the public key. This verifies that the signer has access to the private key and that the data hasn't been modified.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0113

In OSGi, the signer of a bundle is associated with it. With this association, you can grant permissions to a bundle based on its signers. For example, you can assign permissions to all bundles from a particular company, if the company signs its bundles. You can also grant permissions to perform operations on bundles signed by specific principles. These approaches provide a simple yet powerful way to control who can do what inside your application.

Effectively, bundle signing creates a powerful delegation model. An administrator can grant a restricted set of permissions to a signer, after which the signer can create bundles that use those permissions or some subset, without any intervention or communication with the administrator for each particular bundle. To understand how this all fits together, consider the following scenario.

Assume you have a system that features a set of core bundles (which we'll call the *core domain*) and an arbitrary number of third-party plugin bundles (which we'll call the *third-party domain*). This means you expect fully trusted bundles and not completely trusted bundles to exist in your system, but you want to provide a level of isolation between them. Your goal is to create a simple security policy that lets you manage core and third-party bundle domains without knowing the precise bundles in each set. We'll delve into the details of doing this next.

CERTIFICATES AND KEYSTORES

To implement the desired security policy for this scenario, you need to create two root certificates for the core and third-party domains. These root certificates will be used by the framework to establish a chain of trust for the two domains. With these two certificates, you can then sign certificates of core and third-party providers with the appropriate certificate. When they use their individual certificates to sign bundles they've created, the framework can use the root certificates to establish a chain of trust and determine to which domain a bundle belongs.

The details of all this are based on Java 2 JAR file signing, which means the same tools you use to sign JAR files can be used for OSGi. To create the needed certificates and their associated public and private keys, you'll use the `keytool` command provided by the JDK. It can create and manage certificates and keys inside a *keystore*, which is an encrypted file defined by Java for this purpose. For this scenario, you use `keytool` to create two certificates and their associated public/private keys for the core and third-party domains like this:

```
keytool -genkey -keystore keys.ks -alias core -storepass foobar  \
  -keypass barbaz -dname "CN=core,O=baz,C=de"
keytool -genkey -keystore keys.ks -alias third-party \
  -storepass foobar -keypass barbaz -dname "CN=third-party,O=baz,C=de"
```

This creates a keystore called `keys.ks` containing two new key pairs and a certificate for each pair with aliases of `core` and `third-party`. The keystore is protected by the password `foobar`, and the keys themselves have the password `barbaz`. The `-dname` switch allows you to specify the distinguished name you use to identify yourself, which in this case is the `baz` organization in Germany (`de`).

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0114

DISTINGUISHED NAME A standard X.509 structured name, officially identifying a node in a hierarchical namespace. For our purposes, it's sufficient to recognize a distinguished name (DN) as a set of comma-delimited attributes, such as in the example: `CN=core,O=baz,C=de`. These attributes specify the common name, organization, and country, respectively. The hierarchical aspect of this namespace is that it goes from the least significant (but most specific) attribute to the most significant. The root of the tree for these attributes is the country, which is then divided into organization, and further divided into common names within an organization. Order is significant. Two DNs with the same attributes but different order are different DNs.

The next thing to do is sign your key pair certificates with themselves. It may sound a little strange, but this is how you make them root certificates. It's a common thing to do, as you can see by the fact that the `keytool` command has support for it:

```
keytool -selfcert -keystore keys.ks -alias core -storepass foobar \
  -keypass barbaz -dname "CN=core,O=baz,C=de"
keytool -selfcert -keystore keys.ks -alias third-party \
  -storepass foobar -keypass barbaz -dname "CN=third-party,O=baz,C=de"
```

The only difference from the previous command is that you use `-selfcert` instead of `-genkey`.

Now you have key pairs that you can use to sign other certificates or bundles to make them part of your trusted certificate chain. To allow other people to verify your signatures, you need to extract the certificates from the `keys.ks` keystore and import them into a new keystore called `certificates.ks`. Why? Because the `keys.ks` keystore contains your private keys; you need another keystore that contains only your public keys to share with the outside world. Currently, your certificates are saved as key entries (a public/private key pair and its certificate) in the keystore. You need to export them and re-import them as certificate-only entries, which you do like this:

```
keytool -export -v -keystore keys.ks -alias core \
  -file core.cert -storepass foobar -keypass barbaz
keytool -export -v -keystore keys.ks -alias third-party \
  -file third-party.cert -storepass foobar -keypass barbaz
keytool -import -v -keystore certificates.ks -alias core-cert \
  -file core.cert -storepass foobar -keypass barbaz
keytool -import -v -keystore certificates.ks -alias third-party-cert \
  -file third-party.cert -storepass foobar -keypass barbaz
```

You can verify the contents of your keystores like this:

```
> keytool -list -keystore certificates.ks -storepass foobar

third-party-cert, 08.01.2010, trustedCertEntry,
fingerprint (MD5): 15:9B:EE:BE:E7:52:64:D4:9C:C1:CB:5D:69:66:BB:29
core-cert, 08.01.2010, trustedCertEntry,
fingerprint (MD5): CE:37:F8:71:C9:37:12:D0:F1:C8:2B:F9:85:BE:EA:61

> keytool -list -keystore keys.ks -storepass foobar

core, 08.01.2010, PrivateKeyEntry,
fingerprint (MD5): CE:37:F8:71:C9:37:12:D0:F1:C8:2B:F9:85:BE:EA:61
third-party, 08.01.2010, PrivateKeyEntry,
fingerprint (MD5): 15:9B:EE:BE:E7:52:64:D4:9C:C1:CB:5D:69:66:BB:29
```

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0115

You have everything in place now, which means we can look into signing bundles to make them members of one of your domains.

SIGNING BUNDLES

A bundle JAR file can be signed by multiple signers; the signing follows normal Java JAR signing rules. The only additional constraint for a bundle is that all entries inside the bundle must be included in the signature, but entries below the META-INF/ directory aren't included. Normal Java JAR file signing allows for partially signed JAR files, but OSGi doesn't. It's lucky that signing all entries except those below META-INF/ is the default in JAR signing, so you can use the `jarsigner` tool included in the JDK. The following will sign a bundle with your core private key:

```
jarsigner -keystore file:keys.ks \
  -storepass foobar -keypass barbaz core-bundle.jar core
```

Signing another bundle with your third-party private key looks very similar. You specify the appropriate alias:

```
jarsigner -keystore file:keys.ks \
  -storepass foobar -keypass barbaz third-party-bundle.jar third-party
```

For verification, you need the keystore containing the certificates. You can use the `jarsigner` tool for verification as well:

```
jarsigner -verify -keystore file:certificates.ks core-bundle.jar
jarsigner -verify -keystore file:certificates.ks third-party-bundle.jar
```

This command should output `jar verified` if you've correctly signed the bundles. Assuming you have, you now have one bundle in the core domain and one in the third-party domain. This makes it easy for you to grant permissions to either, based on the signer of a bundle, as you'll see next.

14.5.3 BundleSignerCondition

To assign permissions to bundles based on who signed them, you need a condition. The OSGi specification defines the `BundleSignerCondition` for certificate matching, which is specifically based on DN matching. DN matching can seem somewhat complicated, but it needn't be. We'll discuss the details of it shortly; first, let's look at how you construct a `BundleSignerCondition`.

The `BundleSignerCondition` is initialized with a DN matching expression as its first argument and an optional second argument of `!`. If you specify the exclamation mark, it negates the result of the DN matching expression. Consider the following snippet of an encoded `ConditionalPermissionInfo` object containing a `BundleSignerCondition`:

```
ACCEPT {
[org.osgi.service.condpermadmin.BundleSignerCondition "CN=core,O=baz,C=de"]
...
}
```

This matches a bundle on the call stack if it's signed by the core certificate of the example, which means any permissions associated with this entry will be granted to

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0116

the bundle. On the other hand, the following won't match a bundle if it was signed by the core certificate:

```
ACCEPT {
[org.osgi.service.condpermadmin.BundleSignerCondition
  "CN=core,O=baz,C=de" "!"]
...
}
```

The DN matching expression in these two examples illustrates how simple DN matching can be. It can also be sophisticated, because it supports various flavors of wildcard matching. We'll describe that next.

DISTINGUISHED NAME MATCHING DETAILS

You saw that a DN is composed of multiple attributes, like country, organization, and common name. When performing DN matching, you're matching against these attributes using a comma-delimited list, such as what you saw earlier with `CN=core, O=baz,C=de` to match the core certificate. Additionally, because certificates can be signed by other certificates, you can match against the other certificates in the chain: you delimit different certificates with a semicolon.

To match certificates in a chain, use DN matching expressions against the DN associated with each certificate you're trying to match. For example, consider the following DN matching expression:

```
CN=extensions,O=bar,C=fr;CN=core,O=baz,C=de
```

This matches a bundle that was signed by the bar organization from France using its extensions certificate, which was signed by your core certificate. You need to understand two important points about chain matching:

- Matching occurs naturally against the most specific certificate. Certificates further up the chain that aren't mentioned are ignored.
- Order is important, because reversing it indicates the opposite signing relationship.

When you match certificate chains, you're specifying an interest from the most specific certificate of the chain onward.

Both attribute matching and certificate chain matching support wildcards, but the rules for comparison are more complicated than string-based wildcard matching. The different cases are described in table 14.7.

Table 14.7 Certificate DN wildcard matching

Case	Description
Specific attribute wildcards	If a wildcard is used as part of the right-hand argument of an attribute, such as `CN=*,O=baz,C=de` this matches either of the two certificates (core and third-party). You can also use a wildcard for more than one attribute: `CN=*,O=baz,C=*` This matches any certificate from the `baz` organization from any country.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0117

Table 14.7 Certificate DN wildcard matching *(continued)*

Case	Description
Arbitrary attribute wildcards	If a wildcard is used standalone, such as `*,O=baz,C=de` this matches any attributes coming before o and c, regardless of their name or value. For the example, it's another way to match both of your certificates. This kind of wildcard can also be combined with the previous: `*,O=baz,C=*` This also matches all certificates from the `baz` organization from any country.
Certificate chain wildcards	The attribute wildcard can be used in a certificate chain and behaves as described earlier, but when used standalone it matches at most one certificate. For example: `*;CN=core,O=baz,C=de` This matches either a bundle signed by another certificate that was signed by your core certificate or a bundle signed directly by your core certificate. The hyphen wildcard matches zero or more certificates: `-;CN=core,O=baz,C=de` This matches any bundle signed by your core certificate anywhere in the certificate chain.

The rules for certificate matching are also relevant to `AdminPermission`, discussed in section 14.3.3. If you recall, `AdminPermission` accepts an LDAP filter over a limited number of attributes to describe target bundles. The value for the `signer` attribute of the LDAP filter is a DN matching expression.

ESTABLISHING TRUST

In addition to DN matching, a `BundleSignerCondition` will only match if all the certificates in the chain are trusted or are signed by a trusted certificate. Certificates are *trusted* when they're known by the OSGi framework. How do they become known? Prior to OSGi R4.2 specification, this was implementation-specific; but now the standard way is to specify keystores containing trusted certificates using the `org.osgi.framework.trust.repositories` framework configuration property. For example:

```
org.osgi.framework.trust.repositories=\
  /var/trust/keystore.jks:~/.cert/certs.jks
```

The value is a list of file paths, where the paths are separated by the system-specific `File.pathSeparator`. Each file path must point to a JKS keystore, which can't have a password. The framework uses the keystores as trust repositories to authenticate certificates of trusted signers. The stores must be used only as read-only trust repositories to access public keys.

You should now understand how to use certificates to sign your bundles and grant permissions based on the bundle signer. With that out of the way, let's look at how you can use local permissions to know which permissions a bundle needs.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0118

14.6 Local permissions

Bundle signing provides a powerful yet fairly simple mechanism for creating desired security policies. But it doesn't help address one nagging issue: how do you know which permissions to grant a bundle? Even if you've verified a bundle's signature and know that the bundle comes from a trusted provider, you still need to answer this question. Even if you fully trust a provider, it's better to limit a bundle's permissions to a precise set of required permissions to further prevent intended or unintended security breaches.

The standard Java security architecture doesn't help you here; instead, you must rely on prior knowledge about the code's requirements or trial and error. OSGi specifically addresses this issue with a concept called *local permissions*. Local permissions are defined by a resource contained in the bundle, which describes the maximum set of permissions required by the bundle. This set of permission is enforced by the OSGi framework. A bundle can be granted fewer permissions than its local permissions, but it never gets more permissions.

At first blush, it may seem a little odd to have a bundle define its own permissions, but the purpose is more for the deployer to audit and analyze a bundle. Bundles aren't guaranteed to receive the permissions they request and therefore should be programmed to degrade gracefully when they receive less. As a deployer, though, local permissions simplify your life because you can easily determine what permissions are required and which you're willing to give. For example, if the local permissions request the use of network sockets, it's clear that the bundle has the potential to access the wider internet. If you decide this is acceptable, you can trust this audit because it's enforced by the framework at execution time.

What do local permissions look like in practice? The bundle-permission resource is a file in the bundle's OSGI-INF/ directory called permissions.perm. It contains a listing of all of the bundle's required permissions. As a simple example, let's assume you provide a bundle that only wants to export a single package, org.foo. This bundle's OSGi-INF/permissions.perm file is as follows:

```
# Tuesday, Dec 28 2009
# Foo Bundle
( org.osgi.framework.PackagePermission "org.foo" "IMPORT,EXPORT" )
```

Lines that start with a # are comments. All other non-empty lines describe required permissions as encoded `PermissionInfo` objects. This is simple but effective when it comes to auditing the security impact of a given bundle.

You've now learned about some powerful tools for defining a security policy; but in the infamous words of many infomercials, "Wait! There's still more!" In the next section, we'll cover the most advanced tool available: the ability to create custom conditions for your security policy. We'll explore *why* you may want to do this and show you *how* to do it by implementing two custom conditions.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0119

Here you check whether you've already asked the user by looking at the `m_alreadyAsked` flag, which is only necessary to avoid a race condition if multiple threads are trying to set the initial value; after that, the framework will cache the result because the condition is immutable. If the user hasn't already been asked, you create a new `AskTheUser` object with your question ❷ and call its `ask()` method. When you get the result, you set the `alreadyAsked` flag to `true` ❸ to make sure that the user is asked only one time for the given bundle. Finally, you return the result or invert the result if `"!"` was specified in the `ConditionInfo`.

A slightly complicated part of this example is that you need to perform the call to the `ask()` method using `AccessController.doPrivileged()` ❶. This is because the use of Swing will result in a lot of additional permission checks, so you must limit the protection domains involved to the protection domain of the condition itself. Because the condition must be on the class path, it'll have the protection domain of the framework, which needs to have `AllPermission`. If, for whatever reason, you get an exception, you return `false`.

This completes the `AskTheUserCondition` implementation. If you package it as an extension bundle or add it to the framework class path, you can use it to let the user make security decisions by including it in your policy file like this:

```
ACCEPT {
  [ org.foo.AskTheUserCondition "Do you want to allow $symbolic-name to
  provide a shape?" ]
  ( org.osgi.framework.ServicePermission "org.foo.shape.SimpleShape"
  "register")
}
```

We've covered a lot of ground in this chapter, so you should be commended for making it this far. To wrap up the discussion on security, we'll look at an example that pulls everything together.

14.8 Bringing it all back home

What's left is to show you how to start a framework with security enabled. You need to make sure a security manager is installed in the system and tell the framework where it can find the trusted root certificates. You can either set a custom security manager or have the framework install its own security manager. Typically, you need to set the following two framework-configuration properties:

- `org.osgi.framework.security`—Tells the framework to set the security manager
- `org.osgi.framework.trust.repositories`—As mentioned earlier, specifies the repositories of trusted certificates

The `org.osgi.framework.security` property value can be either an empty string or `osgi`. In either case, the framework sets the JVM security manager to an implementation-specific one when started. If the property isn't specified, the framework doesn't set the security manager; security will still work if a security manager is already set, but not all features of OSGi security may work. In particular, if the existing security manager uses `AccessController`, postponed conditions won't work.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0120

Even though some aspects of enabling security are standardized, not all aspects are. As a result, enabling security is handled a little differently by different framework implementations. We'll use the Apache Felix framework to show a concrete example. The Felix framework is special because it provides its Conditional Permission Admin Service implementation as an extension bundle. This means that in addition to setting the previous properties, you also need to deploy the security provider bundle. Luckily, this is easy to do with the bundle launcher; add it to the directory containing the bundles you want to launch.

This gets you a framework with security enabled and an initial security policy. If you want to allow for bundles signed by trusted certificates, you can use the `org.osgi.framework.trust.repositories` property to point to the keystore containing the certificates you trust, but typically a keystore requires the use of a password. Because keystores used with this configuration property can't have passwords, you need to resort to an implementation-specific means to give you what you want. For the Felix framework, you do the following:

```
java -Dorg.osgi.framework.security=osgi \
    -Dfelix.keystore=file:certificates.ks \
    -Dfelix.keystore.pass=foobar \
    -Dfelix.keystore.type=jks \
    -jar launcher.jar bundles
```

In this case, the initial security policy file contains the following:

```
grant { permission java.security.AllPermission; };
```

This sets up your framework with a keystore and the password necessary to access it.

To illustrate what you can do with all of this, let's add to the paint program a security policy that uses the security features you've learned about. The security policy will allow core providers to provide shapes automatically; all others will require explicit approval from the user. Other than provide shape services, bundles are allowed to do anything. Start by creating a policy file with an entry to grant `AllPermission` to bundles signed by the core certificate:

```
ALLOW {
  [ org.osgi.service.condpermadmin.BundleSignerCondition
    "CN=core,O=baz,C=de" ]
   ( java.security.AllPermission "*" "*" )
} "Signed by core"
```

Next, create an entry to grant all other bundles permission to register a shape service based on the condition that the user approves it. For this, use your custom condition like this:

```
ALLOW {
  [ org.foo.condition.ask.AskUserCondition
    "Do you want to allow ${symbolic-name} to provide a shape?" ]
  ( org.osgi.framework.ServicePermission
    "org.foo.shape.SimpleShape" "register" )
} "Ask the user"
```

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0121

If a bundle that isn't signed by your core certificate tries to register a shape service, the user is asked to grant or deny that request. Pretty simple, right? This entry only deals with asking the user to approve shape services from non-core bundles; you still need to create an entry to grant these bundles the ability to do everything else. In that case, you need to grant them `AllPermission` except for registering shape services. You can use a `DENY` access decision, like this:

```
DENY {
  ( org.osgi.framework.ServicePermission
    "org.foo.shape.SimpleShape" "register" )
} "Deny register"
ALLOW {
  ( java.security.AllPermission "*" "*" )
} "All other"
```

All these entries combined form your defined security policy. You may wonder whether the policy is correct. If you follow the rule ordering in the policy file, it looks like it will always prompt the user if a bundle isn't signed by the core. But this isn't the case, because `AskUserCondition` is a postponed condition. That means it's evaluated only if no other rule with an immediate condition implies the permission. Your security policy is evaluated like this:

1. If a bundle is signed by the core certificate, it immediately matches the first rule, which allows it to do anything.
2. If a non-core signed bundle performs any secure operation other than registering a shape service, the first rule doesn't apply, the second rule is postponed, the third rule doesn't apply, and ultimately the fourth rule is matched that allows the bundle to do anything.
3. If a non-core signed bundle tries to register a shape service, the first rule doesn't apply, the second rule is postponed, the third rule applies because the permission is implied, and this rule fails due to the `DENY` access decision. This causes the framework to evaluate the postponed second rule because it logically came before the failed entry, which prompts the user and grants the permission based on the user's reply.

As you can see, you only ask the user at the end if all other rules don't provide the needed permission. The `DENY` access decision of the third rule provides a way to short-circuit the rule evaluation. To see this security policy in action, go into the `chapter14/combined-example/` directory of the book's companion code; type `ant` to build it and the following to run it:

```
java -Dorg.osgi.framework.security=osgi -Djava.security.policy=all.policy \
  -Dfelix.keystore=file:certificates.ks -Dfelix.keystore.pass=foobar  \
  -Dfelix.keystore.type=jks -jar launcher.jar bundles
```

This starts your shell in a security-enabled framework running your security policy. You first need to install the bundles of the paint program:

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0122

```
-> install file:paint-example/shape-4.0.jar
Bundle: 8
-> install file:paint-example/paint-4.0.jar
Bundle: 9
-> start 9
```

These bundles were signed by the core certificate and have `AllPermission`, so you should see an empty paint program (one with no shapes) after starting the paint bundle. Next, install and start the unsigned circle bundle:

```
-> install file:paint-example/circle-4.0.jar
Bundle: 10
-> start 10
```

Doing this causes the user to be prompted to grant the permission, as shown in figure 14.3. If you grant the bundle permission, you get a circle shape in the paint program, as shown in figure 14.4.

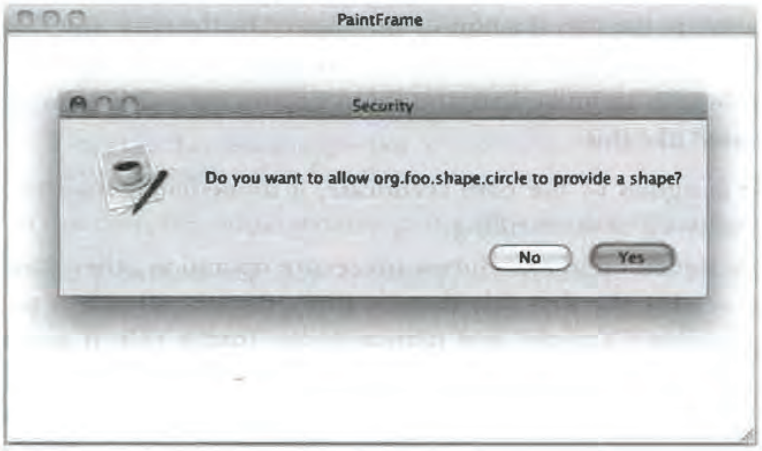


Figure 14.3 Secured paint program prompting the user to grant the unsigned circle bundle permission to provide a shape service



Figure 14.4 Secured paint program after the user has granted permission to the unsigned circle bundle

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0123

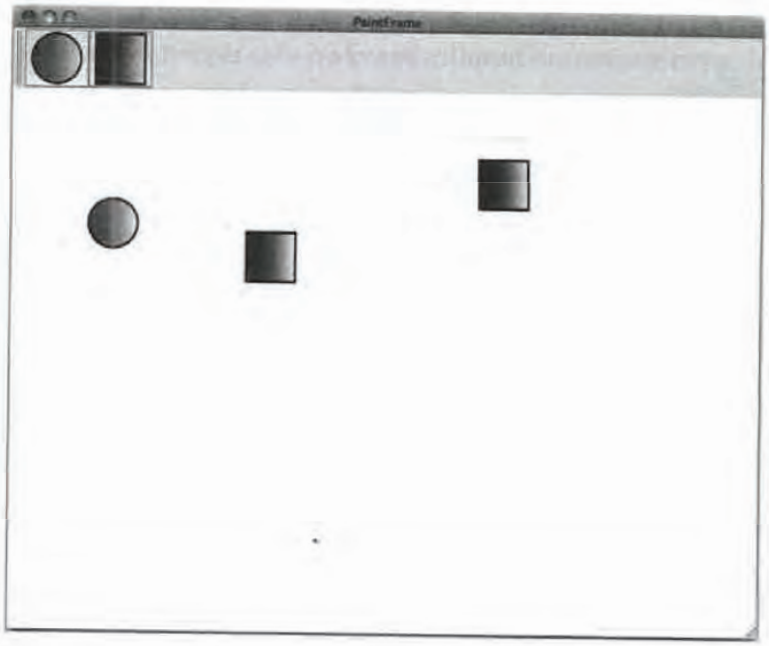


Figure 14.5 Secured paint program with an unsigned circle bundle and a signed square bundle

Finally, to show that core-signed bundles can provide shapes without prompting the user, install and start the square bundle:

```
-> install file:paint-example/square-4.0.jar
Bundle: 11
-> start 11
```

You should now have a paint program with circle and square shapes, as shown in figure 14.5.

Congratulations! If you've made it this far, you know just about everything there is to know about securing your OSGi-based applications. Just do it!

14.9 Summary

In this chapter, we introduced you to the Java security model and showed how OSGi uses it to provide the infrastructure to deploy and manage applications that must run in secure environments. You've learned the following:

- It's important to have security in mind when you're writing bundles, because otherwise they probably won't be good citizens in a security-enabled environment.
- Java security provides the foundation of the OSGi security model.
- OSGi provides the necessary permissions to express security policies for bundles with respect to the three key layers of the OSGi framework: module, lifecycle, and services.
- The Conditional Permission Admin Service introduces a new way of managing security by means of conditions that must be satisfied in order for certain permissions to be applicable.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0124

- You can make your life a lot simpler by signing your bundles with certificates and assigning permissions to bundles based on who signed them.
- Specifying local permission inside of bundles provides a convenient and simple way to audit and enforce the permissions needed by a bundle.
- It's easy to implement and provide custom conditions for use in your security policies.
- Postponed conditions let you defer expensive condition evaluation until the end, which allows you to fine-tune your security policies for efficiency.

With this knowledge under your belt, you can secure your framework according to your specific security policies and develop bundles that can work in security-enabled frameworks. In the next chapter, we'll look into how you can use and provide web services in OSGi as well as how to build web applications on top of OSGi.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0125

JAVA

LC CIP
0 028 813 668 3

OSGi IN ACTION

Hall • Pauls • McCulloch • Savage

Free eBook
SEE INSERT

OSGi is a Java-based framework for creating applications as a set of interconnected modules. OSGi lets you install, start, stop, update, or uninstall modules at execution time without taking down your entire system. It's the backbone of the Eclipse plugin system, as well as many Java EE containers, such as GlassFish, Geronimo, and WebSphere.

OSGi in Action provides a clear introduction to OSGi concepts with examples that are relevant both for architects and developers. You'll start with the central ideas of OSGi: bundles, module lifecycles, and interaction among application components. With the core concepts well in hand, you'll explore numerous application scenarios and techniques. You'll learn how to migrate legacy systems to OSGi and how to test, debug, and manage applications.

What's Inside

- Core ideas of OSGi
- Vocabulary, tools, and strategies
- Applying OSGi

This book assumes readers with a working knowledge of Java, but requires no previous exposure to OSGi.

Richard S. Hall, **Karl Pauls**, **Stuart McCulloch**, and **David Savage** are all respected Java developers and committers on the Apache Felix OSGi implementation.

For access to the book's forum and a free ebook for owners of this book, go to manning.com/OSGiinAction

"An impressive book."
—From the foreword by Peter Kriens OSGi Technical Director

"A lucid explanation of an intricate topic."
—John S. Griffin, Overstock.com

"Easy to read ... explains everything you need to know."
—Jason Lee, Oracle

"Straight from the experts."
—Erik Van Oosten, JTeam

"Hit the ground running with this book."
—David Dossot Coauthor of *Mule in Action*

ISBN-13: 978-1-933988-91-7
ISBN-10: 1-933988-91-6
54999
9 781933 988917

MANNING $49.99 / Can $61.99 [INCLUDING eBOOK]

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0126

Appendix CV

June A. Munford
Curriculum Vitae

Education

University of Wisconsin-Milwaukee - MS, Library & Information Science, 2009
Milwaukee, WI

- Coursework included cataloging, metadata, data analysis, library systems, management strategies and collection development.
- Specialized in library advocacy, cataloging and public administration.

Grand Valley State University - BA, English Language & Literature, 2008
Allendale, MI

- Coursework included linguistics, documentation and literary analysis.
- Minor in political science with a focus in local-level economics and government.

Professional Experience

Researcher / Expert Witness, October 2017 – present
Freelance ● Pittsburgh, Pennsylvania & Grand Rapids, Michigan

- Material authentication and public accessibility determination. Declarations of authenticity and/or public accessibility provided upon research completion. Experienced with appeals and deposition process.

- Research provided on topics of public library operations, material publication history, digital database services and legacy web resources.

- Past clients include Alston & Bird, Arnold & Porter, Baker Botts, Fish & Richardson, Erise IP, Irell & Manella, O'Melveny & Myers, Perkins-Coie, Pillsbury Winthrop Shaw Pittman and Slayden Grubert Beard.

Library Director, February 2013 - March 2015
Dowagiac District Library ● Dowagiac, Michigan

- Executive administrator of the Dowagiac District Library. Located in

Southwest Michigan, this library has a service area of 13,000, an annual operating budget of over $400,000 and total assets of approximately $1,300,000.

● Developed careful budgeting guidelines to produce a 15% surplus during the 2013-2014 & 2014-2015 fiscal years while being audited.

● Using this budget surplus, oversaw significant library investments including the purchase of property for a future building site, demolition of existing buildings and building renovation projects on the current facility.

● Led the organization and digitization of the library's archival records.

● Served as the public representative for the library, developing business relationships with local school, museum and tribal government entities.

● Developed an objective-based analysis system for measuring library services - including a full collection analysis of the library's 50,000+ circulating items and their records.

November 2010 - January 2013
Librarian & Branch Manager, Anchorage Public Library ● Anchorage, Alaska

● Headed the 2013 Anchorage Reads community reading campaign including event planning, staging public performances and creating marketing materials for mass distribution.

● Co-led the social media department of the library's marketing team, drafting social media guidelines, creating original content and instituting long-term planning via content calendars.

● Developed business relationships with The Boys & Girls Club, Anchorage School District and the US Army to establish summer reading programs for children.

June 2004 - September 2005, September 2006 - October 2013
Library Assistant, Hart Area Public Library
Hart, MI

● Responsible for verifying imported MARC records and original MARC

cataloging for the local-level collection as well as the Michigan Electronic Library.

● Handled OCLC Worldcat interlibrary loan requests & fulfillment via ongoing communication with lending libraries.

Professional Involvement

Alaska Library Association - Anchorage Chapter
- Treasurer, 2012

Library Of Michigan
- Level VII Certification, 2008
- Level II Certification, 2013

Michigan Library Association Annual Conference 2014
- New Directors Conference Panel Member

Southwest Michigan Library Cooperative
- Represented the Dowagiac District Library, 2013-2015

Professional Development

Library Of Michigan Beginning Workshop, May 2008
Petoskey, MI
- Received training in cataloging, local history, collection management, children's literacy and reference service.

Public Library Association Intensive Library Management Training, October 2011
Nashville, TN
- Attended a five-day workshop focused on strategic planning, staff management, statistical analysis, collections and cataloging theory.

Alaska Library Association Annual Conference 2012 - Fairbanks, February 2012
Fairbanks, AK
- Attended seminars on EBSCO advanced search methods, budgeting, cataloging, database usage and marketing.

Depositions

2019 ● Fish & Richardson

Apple v. Qualcomm (IPR2018-001281, 39521-00421IP, IPR2018-01282 and 39521-00421IP2)

2019 ● Erise IP

Implicit, LLC v. Netscout Systems, Inc (Civil Action No. 2:18-cv-53-JRG)

2019 ● Perkins-Coie

Adobe Inc. v. RAH Color Technologies LLC (Cases IPR2019-00627, IPR2019-00628, IPR2019-00629 and IPR2019-00646)

2020 ● O'Melveny & Myers

Maxell, Ltd. v. Apple Inc. (Case No. 5:19-cv-00036-RWS)

2021 ● Pillsbury Winthrop Shaw Pittman LLP

Intel v. SRC (IPR2020-1449)

2022 ● Perkins-Coie

Realtek v. Future Link (IPR2021-01182)

2023 ● Fish & Richardson

Neuroderm Ltd. v. Abbvie, Inc (Case No. PGR2022-00040)

2023 ● Fish & Richardson

Nearmap US Inc. v. Pictometry International Corp. (IPR2022-00735)

2023 ● Fish & Richardson

Samsung Electronics v. MemoryWeb LLC (Case No. 39843-0136PS1)

2023 ● Pillsbury Winthrop Shaw Whitman LLP

Gravel Rating Systems v. Costco Wholesale Corp. (Civil Action No. 4:21-cv-149-ALM)

2024 ● Willkie-Farr

Netflix, Inc. v. VideoLabs. Inc. (IPR2023-00628)

Limited Case and Clientele History

Alston & Bird

- Ericsson

  v. Collision Communications (IPR2022-01233)

- Nokia

  v. Neptune Subsea, Xtera (Case No. 1:17-cv-01876)

- Universal Electronics Inc

  v. Roku Inc (IPR2022-00818)

Arnold & Porter

- Ivantis

  v. Glaukos (Case No. 8:18-cv-00620)

- Samsung

  v. Jawbone (Case No. 2:21-cv-00186)

Benesch Friedlander Coplan & Aronoff

- Voyis

  v. Cathx (Case No. 5:21-cv-00077-RWS)

Deschert LLP

- Smaxtec Animal Care

  v. ST Reproductive Technologies, LLC (IPR2024-00885)

Erise I.P.

- Apple

  v. Ericsson Inc. (IPR2022-00715)

  v. Future Link Systems (IPRs 6317804, 6622108, 6807505, and 7917680)

  v. INVT (Case No. 20-1881)

  v. Navblazer LLC (IPR2020-01253)

  v. Qualcomm (IPR2018-001281, 39521-00421IP, IPR2018-01282, 39521-00421IP2)

  v. Quest Nettech Corp (Case No. 2:19-cv-00118-JRG)

  v. Telefonaktiebolaget LM Ericsson (IPR2022-00275)

  v. Theta IP, LLC (IPR2024-00818)

- Fanduel

  v. CGT (Case No. 19-1393)

- Garmin

  v. Phillips North America LLC (Case No. 2:19-cv-6301-AB-KS)

- Netscout
    - v. Longhorn HD LLC (Case No. 2:20-cv-00349)
    - v. Implicit, LLC (Civil Action No. 2:18-cv-53-JRG)

- Sony Interactive Entertainment LLC
    - v. Bot M8 LLC (IPR2020-01288)
    - v. Infernal Technology LLC (Case No. 2:19-CV-00248-JRG)

- Unified Patents
    - v. GE Video Compression (Civil Action No. 2:19-cv-248)

Fish & Richardson

- Apple
    - v. AliveCor (3:21-cv-03958)
    - v. LBS Innovations (Case No. 2:19-cv-00119-JRG-RSP)
    - v. Koss Corporation (IPR2021-00305)
    - v. Masimo (IPR 50095-0012IP1, 50095-0012IP2, 50095-0013IP1, 50095-0013IP2, 50095-0006IP1, 50095-0135IP1)
    - v. Neonode (Case No. 21-cv-08872-EMC)
    - v. Qualcomm (IPR2018-001281, 39521-00421IP, IPR2018-01282, 39521-00421IP2)

- Dell
    - v. Neo Wireless (IPR2022-00616)

- Dish Network
    - v. Entropic Communications, LLC (Case No. 2:2023-CV-01043)

v. Realtime Adaptive Streaming (Case No. 1:17-CV-02097-RBJ)

v. TQ Delta LLC (Case No. 18-1798)

- Evapco Dry Cooling

  v. SPG Dry Cooling (IPR2021-00688)

- Genetec

  v. Sensormatic Electronics (Case No. 1:20-CV-00760)

- Huawei

  v. Bell Northern Research LLC (IPR2019-01174)

- Kianxis

  v. Blue Yonder (Case No. 3:20-cv-03636)

- LG Electronics

  v. Bell Northern Research LLC (Case No. 3:18-cv-2864-CAB-BLM)

- Metaswitch

  v. Sonus Networks (IPR2018-01719)

- Microsoft

  v. Throughputer Inc (IPR2022-00757)

- Mom Enterprises

  v. Ddrops Company (Case No. 1:22-cv-00332)

- MLC Intellectual Property

v. MicronTech (Case No. 3:14-cv-03657-SI)

- Nearmap Inc
  - v. EagleView Technologies (IPR2022-01009)

- Neuroderm Ltd.
  - v. Abbvie, Inc (Case No. PGR2022-00040)

- Realtek Semiconductor
  - v. Future Link (IPR2021-01182)

- Quectel
  - v. Koninklijke Philips (Case No. 1:20-cv-01710)

- Samsung
  - v. Aire Technology (IPR2022-00877)
  - v. Bell Northern Research (Case No. 2:19-cv-00286-JRG)
  - v. Communication Technologies Inc (IPR2022-01221)
  - v. Jawbone Innovations (IPR2022-00865)
  - v. MemoryWeb LLC (IPR2022-00885)
  - v. Dodots Licensing Solutions (IPR2023-00701)
  - v. Telefonaktiebolaget LM Ericsson (IPR2021-00615)

- Texas Instruments
  - v. Vantage Micro LLC (IPR2020-01261)

- Xilinx
  - v. Sentient Sensors LCC (Case No. 1:22-cv-00173)

Groombridge Wu

- Nearmap

  v. Eagle View Technologies Inc. and Pictometry International Corp. (Case No. 2:21-cv-00283-TS-DAO)

Irell & Manella

- Curium

O'Melveny & Myers

- Apple

  v. Maxell (Case No. 5:19-cv-00036-RWS)

- Disney Media & Entertainment Distribution LLC

  v. Digital Media Technology Holdings, LLC (Case No. 2:22-cv-01642)

- Micron Technology Inc.

  v. Besang Inc (IPR2023-00900)

- Samsung

  v. Daedalus Prime LLC (1335 Investigation)

Perkins-Coie

- Amazon

  v. Nokia (IPR2024-00847 and IPR2024-00848)

- Heru Industries

  v. The UAB Research Foundation (IPR2022-01148)

- Intel Corporation

  v. BeSang Inc. (IPR2023-00991)

  v. BitMICRO (Case No. 5:23-cv-00625)

  v. Telefonaktiebolaget L M Ericsson (IPR2024-00610)

- Realtek Semiconductor

  v. Future Link (IPR2021-01182)

- Twitter Inc

  v. VOIP-Pal.com (Case No. 3:20-cv-02397-JD)

- TCL Industries

  v. Koninklijke Philips NV (IPR2021-00495, IPR2021-00496 and IPR2021-00497)

Pillsbury Winthrop Shaw Pittman

- Intel

  v. FG SRC LLC (Case No. 6:20-cv-00315)

- Gravel Rating Systems

  v. Costco (Case No. 4:21-cv-149)
  v. Lowe's Home Centers (Case No. 4:21-cv-150)
  v. T-Mobile USA (Case No. 4:21-cv-152)
  v. Kohl's Inc. (Case No. 4:21-cv-258)
  v. Under Armor (Case No. 4:21-cv-356)

Quinn Emanuel

- ServiceNow, Inc.

  v. InQuisient, Inc. (Case No. 22-900-CJB)

- Wolfspeed

  v. The Trustees of Purdue University (Case No. 1:2021-cv-00840)

Sheppard, Mullin, Richter & Hampton LLP

- Cadence Design Systems Inc.

  v. Semiconductor Design Technologies (Case No. 3:23-cv-01001)

Stern, Kessler, Goldstein & Fox

- Preliminary Research

Willkie, Farr & Gallagher LLP

- Neurent

  v. The Foundry, LLC (IPR2024-00669)

- Netflix, Inc.

  v. VideoLabs, Inc. (IPR2023-00628)

Appendix B

MARC 21 Reference Materials

Part VII: A Summary of Commonly Used MARC 21 Fields
Part VIII: A List of Other Fields Often Seen in MARC Records
Part IX: The Leader
Part X: Field 008 for Books

Part VII:

A Summary of Commonly Used MARC 21 Fields

This is a summary of the MARC 21 tags used most frequently by libraries in entering their own bibliographic records. For full listings of all MARC 21 tags, indicators, and subfield codes, see *MARC 21 Format for Bibliographic Data*.

In the explanations on these pages:

Tags -- The tags (3-digit numbers) are followed by the names of the fields they represent. In this summary, and in the *MARC 21 Format for Bibliographic Data*, if a tag can appear more than once in one bibliographic record, it is labeled repeatable (R). If it can only be used once, it is labeled non-repeatable (NR). For example, a catalog record can have several subjects, so the tags for subject added entries (6XX) are labeled repeatable (R).

Indicators -- The use of indicators is explained in fields where they are used. Indicators are one-digit numbers. Beginning with the 010 field, in every field -- following the tag -- are two character positions, one for Indicator 1 and one for Indicator 2. The indicators are not actually defined in all fields, however. And it is possible that a 2nd indicator will be used, while the 1st indicator remains undefined (or vice versa). When an indicator is undefined, the character position will be represented by the character # (for blank space).

Subfield codes -- All the data in each field (beginning with the 010 field) is divided into subfields, each of which is preceded by a delimiter-subfield code combination. The most common subfield codes used with each tag are shown. Each subfield code is preceded by the character $, signifying a delimiter. The name of the subfield follows the code.

In general, every field MUST have a subfield 'a' ($a). One exception that is often seen is in Field 020 (ISBN), when the ISBN information (subfield **$a**) is unavailable but the price (subfield **$c**) is known. Some subfields are repeatable. In this summary, repeatability is noted for only the more common repeatable subfields.

Examples: Examples follow the explanation for each field. For clarity, one space has been placed between the tag and the first indicator, one space has been placed between the second indicator and the first delimiter- subfield code, and one space has been inserted between the delimiter-subfield code and the subfield data.

010 Library of Congress Control Number -- (LCCN)
(NR, or Not Repeatable

Indicators undefined.
Subfield used most often:
$a -- Library of Congress control number

```
Example:    010 ##  $a ###86000988#
```

020 International Standard Book Number -- (ISBN)
(R, or Repeatable)

Indicators undefined.
Subfields used most often:
$a -- International Standard Book Number
$c -- Terms of availability (often a price)
$z -- Cancelled/invalid ISBN (R)

```
Example:    020 ##  $a 0877547637
```

040 Cataloging source -- (NR)

Indicators undefined.
Subfields used most often:
$a -- Original cataloging agency
$c -- Transcribing agency
$d -- Modifying agency (R)

```
Example:    040 ## $a DLC
                   $c DLC
                   $d gwhs
```

100 Main entry -- Personal name -- (primary author)
(NR; there can be only one main entry)

Indicator 1: Type of personal name entry element
0 -- Forename
1 -- Surname (this is the most common form)
3 -- Family name
Indicator 2 undefined.
Indicator 2 became obsolete in 1990. Older records may display 0 or 1
Subfields used most often:
$a -- Personal name
$b -- Numeration
$c -- Titles and other words associated with a name (R)
$q -- Fuller form of name
$d -- Dates associated with a name (generally, year of birth)

```
Example:    100 1# $a Gregory, Ruth W.
                   $q (Ruth Wilhelme),
                   $d 1910-
```

130 Main entry -- Uniform title -- (NR)

Indicator 1: Nonfiling characters
0-9 -- Number of nonfiling characters present (for initial articles, including spaces)
Indicator 2 undefined.
Indicator 2 became obsolete in 1990. (See 100 above.)
Subfields used most often:
$a -- Uniform title
$p -- Name of part/section of a work (R)
$l -- Language of a work
$s -- Version
$f -- Date of a work

```
Example:    130 0# $a Bible.
                   $p O.T.
                   $p Psalms.
```

240 Uniform title (NR)

Indicator 1: Uniform title printed or displayed
0 -- Not printed or displayed
1 -- Printed or displayed (most common)
Indicator 2: Nonfiling characters
0-9 -- Number of nonfiling characters present (for initial articles, including spaces)
Subfields used most often:
$a -- Uniform title
$l -- Language of a work
$f -- Date of a work

```
Example:    240 10 $a Ile mystérieuse.
                   $l English,
                   $f 1978
```

245 Title Statement (NR)

Indicator 1: Title added entry
(Should the title be indexed as a title added entry?)
0 -- No title added entry
(indicates a title main entry; i.e. no author is given)
1 -- Title added entry
(the proper indicator when an author given in 1XX; the most common situation)
Indicator 2: Nonfiling characters
0-9 -- Number of nonfiling characters present, including spaces; usually set at zero, except when the title begins with an article; e.g., for *The robe*, the second indicator would be set to 4. The letters *T, h, e,* and the space following them are then ignored in alphabetizing titles. The record will be automatically filed under "*r*" -- for *Robe.*
Subfields used most often:
$a -- Title proper
$h -- Medium (often used for non-book media)
$p -- Name of part/section of a work (R)
$b -- Reminder of title (subtitles, etc.)
$c -- Remainder of title page transcription/Statement of responsibility

```
Example:    245 14 $a The DNA story :
                   $b a documentary history of gene
                      cloning /
                   $c James D. Watson, John Tooze.
```

246 Varying form of title (R)

Indicator 1: Note/title added entry controller
1 -- Note, title added entry
3 -- No note, title added entry
Indicator 2: Type of title
-- No information provided
0 -- Portion of title
1 -- Parallel title
4 -- Cover title
8 -- Spine title
Subfield used most often:
$a -- Title proper

```
Example:    246 3# $a Four corners power review
```

250 Edition statement (NR)

Indicators undefined.
Subfield used most often:
$a -- Edition statement

```
Example:    250 ## $a 6th ed.
```

260 Publication, distribution, etc. (Imprint) (R)

Indicator 1: Sequence of publishing statements
-- No information provided
Indicator 2: Undefined
Subfields used most often:
$a -- Place of publication, distribution, etc. (R)
$b -- Name of publisher, distributor, etc. (R)
$c -- Date of publication, distribution, etc. (R)

```
Example:    260 ## $a New York :
                   $b Chelsea House,
                   $c 1986.
```

300 Physical description (R)

Indicators undefined.
Subfields used most often:
$a -- Extent (number of pages) (R)
$b -- Other physical details (usually illustration information)
$c -- Dimensions (cm.) (R)
$e -- Accompanying material (for example, "teacher's guide" or "manual")

```
Example:    300 ## $a 139 p. :
                   $b ill. ;
                   $c 24 cm.
```

440 Series statement / Added entry--Title

This field was made obsolete in 2008 to simplify the series statement. See 490 and 830.

490 Series statement (No added entry is traced from field) (R)

Indicator 1: Specifies whether series is traced (whether an 8XX tag is also present)
0 -- Series not traced
1 -- Series traced (8XX is in record)
Indicator 2 undefined.
Subfield used most often:
$a -- Series statement (R)
$v -- Volume number (R)

```
Example:    490 1# $a Colonial American craftsmen
```

500 General note (R)

Indicators undefined.
Subfield used most often:
$a -- General note (Used when no specialized note field has been defined for the information. Examples: Notes regarding the index; the source of the title; variations in title; descriptions of the nature, form, or scope of the item.)

```
Example:    500 ##  $a Includes index.
```

504 Bibliography, etc. note (R)

Indicators undefined.
Subfield used most often:
$a -- Bibliography, etc. note

```
Example:    504 ## $a Includes bibliographical
                    references.
```

505 Formatted contents note (R)

Indicator 1: Type of contents note
0 -- Complete contents
1 -- Incomplete contents (used with multivolume set when some volumes are not yet published)
2 -- Partial contents
Indicator 2: Level of content designation
-- Basic
Subfield used most often:
$a -- Formatted contents note

```
Example:    505 0# $a Pride and prejudice -- Emma
                    -- Northanger Abbey.
```

520 Summary, etc. note (R)

Indicator 1: Display constant controller
-- Summary
1 -- Review
2 -- Scope and content
3 -- Abstract
Indicator 2 undefined
Subfields used most often
$a -- Summary, abstract, or annotation
$b -- Expansion of summary note

```
Example:    520 ## $a This basic guide to parliamentary
                    procedure tells how to conduct
                    and participate in a meeting
                    properly.
```

600 Subject added entry -- Personal name (R)

Indicator 1: Type of personal name entry element
0 -- Forename
1 -- Surname (this is the most common form)
3 -- Family name
Indicator 2: Subject heading system/thesaurus (identifies the specific list or file which was used)
0 -- Library of Congress Subject Headings
1 -- LC subject headings for children's literature
2 -- Medical Subject Headings
3 -- National Agricultural Library subject authority file
4 -- Source not specified
5 -- Canadian Subject Headings
6 -- Répertoire de vedettes-matière
7 -- Source specified in subfield $2

(Note regarding Sears subject headings: The MARC 21 format does not provide an assigned indicator for Sears subject headings. Therefore, an indicator of 7 is used, and the MARC defined code "sears" is placed in subfield $2.)

Subfields used most often:
$a -- Personal name (surname and forename)
$b -- Numeration
$c -- Titles and other words associated with a name (R)
$q -- Fuller form of name
$d -- Dates associated with a name (generally, year of birth)
$t -- Title of a work

$v -- Form subdivision (R)
$x -- General subdivision (R)
$y -- Chronological subdivision (R)
$z -- Geographic subdivision (R)
$2 -- Source of heading or term (used with 2nd indicator of 7)

```
Example:    600 10 $a Shakespeare, William,
                   $d 1564-1616
                   $x Comedies
                   $x Stage history.
```

```
Example:    600 10 $a Shakespeare, William,
                   $d 1564-1616
                   $x Knowledge
                   $z Rome
                   $v Congresses.
```

Notice that subfields $v, $x, and $z in the 600 field are repeatable. Subfields $v, $x, $y, and $z do not have to be in alphabetical order. They will be in the order prescribed by the instructions given by the subject heading system.

610 Subject added entry -- Corporate name (R)

Indicator 1: Type of corporate name entry element
0 -- Inverted name (not used with AACR2)
1 -- Jurisdiction name
2 -- Name in direct order
Indicator 2: Subject heading system/thesaurus.
See indicator 2 under 600
Subfields used most often:
$a -- Corporate name or jurisdiction name as entry element
$b -- Subordinate unit (R)
$v -- Form subdivision (R)
$x -- General subdivision (R)
$y -- Chronological subdivision (R)
$z -- Geographic subdivision (R)
$2 -- Source of heading or term (used with 2nd indicator of 7)

```
Example:    610 10 $a United States.
                   $b Army Air Forces
                   $v Biography.
```

650 Subject added entry -- Topical term (Most subject headings fit here.) (R)

Indicator 1: Level of subject
-- No information provided
Indicator 2: Subject heading system/thesaurus
(identifies the specific list or file which was used)
0 -- Library of Congress Subject Headings
1 -- LC subject headings for children's literature
2 -- Medical Subject Headings
3 -- National Agricultural Library subject authority file
4 -- Source not specified
5 -- Canadian Subject Headings
6 -- Répertoire de vedettes-matière
7 -- Source specified in subfield $2

Note regarding Sears subject headings: The MARC 21 format does not provide an assigned indicator for Sears subject headings. Therefore, an indicator of 7 is used, and the MARC defined code "sears" is placed in subfield $2.)

Subfields used most often:
$a -- Topical term
$v -- Form subdivision (R)
$x -- General subdivision (R)
$y -- Chronological subdivision (R)
$z -- Geographic subdivision (R)
$2 -- Source of heading or term used with 2nd indicator of 7)

```
Example:    650 #0 $a Theater
                   $z United States
                   $v Biography
                   $v Dictionaries.
```

Notice that subfields $v, $x, and $z in the 650 field are repeatable. Subfields $v, $x, $y, and $z do not have to be in alphabetical order. They will be in the order prescribed by the instructions given by the subject heading system.

651 Subject added entry -- Geographic name (R)

Indicator 1: undefined.
Indicator 2: Subject heading system/thesaurus.
See indicator 2 under 600
Subfields used most often:
$a -- Geographic name
$v -- Form subdivision (R)
$x -- General subdivision (R)
$y -- Chronological subdivision (R)
$z -- Geographic subdivision (R)
$2 -- Source of heading or term (used with 2nd indicator of 7)

```
Example:    651 #0 $a United States
                   $x History
                   $v Chronology.
```

Notice that subfields $v, $x, and $z in the 651 field are repeatable. Subfields $v, $x, $y, and $z do not have to be in alphabetical order. They will be in the order prescribed by the instructions given by the subject heading system.

700 Added entry -- Personal name (R)

Indicator 1: Type of personal name entry element
0 -- Forename
1 -- Surname (this is the most common form)
3 -- Family name
Indicator 2: Type of added entry
-- No information provided (most common; co-authors, editors, etc.)
2 -- Analytical entry (The values for Indicator 2 changed in 1994 with Format Integration, and older records may display additional values. An analytical entry involves an author/title of an item contained in a work.)
Subfields used most often:
$a -- Personal name
$b -- Numeration
$c -- Titles and other words associated with a name (R)
$q -- Fuller form of name
$d -- Dates associated with a name (generally, year of birth)
$e -- Relator term (such as ill.) (R)
$4 -- Relator code (R)

```
Example:    700 1# $a Baldridge, Letitia.
```

710 Added entry -- Corporate name (R)

Indicator 1: Type of corporate name entry element
0 -- Inverted name (not used with AACR2)
1 -- Jurisdiction name
2 -- Name in direct order
Indicator 2: Type of added entry.
See Indicator 2 under 700
-- No information provided
2 -- Analytical entry
Subfields used most often:
$a -- Corporate name or jurisdiction name as entry element
$b -- Subordinate unit (R)

```
Example:    710 2# $a Sunburst Communications (Firm)
```

740 Added entry -- Uncontrolled related/analytical title (R)

Indicator 1: Nonfiling characters
0-9 -- Number of nonfiling characters present (for initial articles, including spaces)
Indicator 2: Type of added entry. See Indicator 2 under 700
-- No information provided
2 -- Analytical entry
(This field was redefined in 1994 with Format Integration. Prior to 1994, the field was also used for variant titles, such as a different wording on a spine title. In records created since Format Integration, those variant tiles appear in a 246 field.)
Subfield used most often:
$a -- Title

```
Example:    740 02 $a Uncle Vanya.
```

800 Series added entry -- Personal name (R)

Indicator 1: Type of personal name entry element
0 -- Forename
1 -- Surname

3 -- Family name

Indicator 2 undefined.

Subfields used most often:

$a -- Personal name
$b -- Numeration
$c -- Titles and other words associated with a name (R)
$q -- Fuller form of name
$d -- Dates associated with a name (generally, year of birth)
$t -- Title of a work (the series)
$v -- Volume number

```
Example:    800 1# $a Fisher, Leonard Everett.
                  $t Colonial American craftsmen.
```

830 Series added entry -- Uniform title (R)

Indicator 1 undefined.

Indicator 2: Nonfiling characters

0-9 -- Number of nonfiling characters present (for initial articles, including spaces)

Subfield used most often:

$a -- Uniform title
$v -- Volume number

```
Example:    830 #0 $a Railroads of America (Macmillan)
```

[Back to Top of Page]

Part VIII:

A List of Other Fields Often Seen in MARC Records

001	Control number
003	Control number identifier
005	Date and time of latest transaction
006	Fixed-length data elements -- additional material characteristics
007	Physical description fixed field
008	Fixed length data elements (See Part X)
022	International Standard Serial Number (ISSN)
037	Source of acquisition
041	Language code
043	Geographic area code
050	Library of Congress call number
060	National Library of Medicine call number
082	Dewey Decimal classification number (the one recommended by the Library of Congress; locally-assigned call numbers may appear elsewhere)
110	Main entry -- Corporate name (less frequent under AACR2 rules)
256	Computer file characteristics
263	Projected publication date (indicates a CIP -- Cataloging in Publication -- record)
306	Playing time
508	Creation/production credits note
510	Citation/references note (review sources)
511	Participant or performer note
521	Target audience note (first indicator: 0 = reading grade level, 1 = interest age level, 2 = interest grade level, 3 = special audience characteristics, 4 = motivation interest level)
530	Additional physical form available note
538	System details note
586	Awards note
656	Index term -- Occupation
730	Added entry -- Uniform title
852	Location
856	Electronic location and access
9XX	Reserved for local use. (They are used by vendors, systems, or individual libraries to exchange additional data)

[Back to Top of Page]

Part IX:

The Leader

There are 24 positions in the Leader, numbered from 00 to 23. For fuller explanation, see the *MARC 21 Format for Bibliographic Data.*

00-04 Record length (calculated by the computer for each record)

05 Record status

a = increase in encoding level
c = corrected or revised

d = deleted
n = new
p = increase in encoding from prepublication (previous CIP)

06 Type of record
a = language material
c = printed music
d = manuscript music
e = cartographic material
f = manuscript cartographic material
g = projected medium
i = nonmusical sound recording
j = musical sound recording
k = 2-dimensional nonprojectable graphic
m = computer file
o = kit
p = mixed materials
r = 3-dimensional artifact or naturally occurring object
t = manuscript language material

07 Bibliographic level
a = monographic component part
b = serial component part
c = collection
d = subunit
i = integrating resource
m = monograph/item
s = serial

08 Type of control
= no specified type
a = archival

09 Character coding scheme
= MARC-8
a = UCS/Unicode

10 Indicator count (always "2")

11 Subfield code count (always "2")

12-16 Base address of data (calculated by the computer for each record)

17 Encoding level
= full level
1 = full level, material not examined
2 = less-than-full level, material not examined
3 = abbreviated level
4 = core level
5 = partial (preliminary) level
7 = minimal level
8 = prepublication level (CIP)
u = unknown
z = not applicable

18 Descriptive cataloging form
= non-ISBD
a = AACR2
i = ISBD
u = unknown

19 Multipart resource record level
= Not specified or not applicable
a = Set
b = Part with independent title
c = Part with dependent title

20 Length of the length-of-field portion (always "4")

21 Length of the starting-character-position portion (always "5")

22 Length of the implementation-defined portion (always "0")

23 Undefined (always "0")

[Back to Top of Page]

Part X:

Field 008 for Books

Field 008 is used for Fixed Length Data Elements ("Fixed Field Codes"). There are 40 character positions in field 008, numbered from 00-39. Undefined positions must contain either a blank (#) or a fill character (|). Positions 00-17 and 35-39 are defined the same way for all media.

The information shown here for positions 18-34 applies only to books. For explanation of all the positions below and for positions 18-34 for other media, see the *MARC 21 Format for Bibliographic Data*.

Note that field 008 has no indicators or subfield codes.

00-05 Date entered on file (YYMMDD),
where Y=year, M=month, and D=day

06 Type of date/publication status:
b = no dates given; B.C. date involved
e = detailed date
s = single known date/probable date
m = multiple dates
r = reprint/reissue date (Date 1) and original date (Date 2)
n = dates unknown
q = questionable date
t = publication date and copyright date
| = no attempt to code

07-10 Date 1/beginning date of publication

11-14 Date 2/ending date of publication

Date fields contain the year(s) of publication. The type of date(s) in these elements are specified in fixed field element 06: Type of date/publication status. (For further details, see the field 008 description in the *MARC 21 Format for Bibliographic Data*.)

15-17 Place of publication, production, or execution
For example:
pk# = Pakistan
cau = California (US)

(For a full list of codes used in these positions, see the *MARC Code List for Countries*.)

18-21 Illustrations (up to 4 codes):
= no illustrations
a = illustrations
b = maps
c = portraits
d = charts
e = plans
f = plates
g = music
h = facsimiles
i = coats of arms
j = genealogical tables
k = forms
l = samples
m = phonodisc, phonowire, etc.
o = photographs
p = illuminations
| = no attempt to code

22 Target audience:
= unknown or not specified
a = preschool
b = primary
c = pre-adolescent
d = adolescent
e = adult
f = specialized
g = general
j = juvenile
| = no attempt to code

23 Form of item:
= none of the following
a = microfilm
b = microfiche
c = microopaque
d = large print
f = braille
r = regular print reproduction
s = electronic
| = no attempt to code

24-27 Nature of contents (up to 4):
= no specified nature of contents
a = abstracts/summaries
b = bibliographies (is one or contains one)
c = catalogs
d = dictionaries
e = encyclopedias
f = handbooks
g = legal articles
i = indexes
j = patent document
k = discographies
l = legislation
m = theses
n = surveys of literature
o = reviews
p = programmed texts
q = filmographies
r = directories
s = statistics
t = technical reports
u = standards/specifications
v = legal cases and notes
w = law reports and digests
z = treaties
| = no attempt to code

28 Government publication:
= not a government publication
i = international intergovernmental
f = federal/national
a = autonomous or semi-autonomous component
s = state, provincial, territorial, dependent, etc.
m = multistate
c = multilocal
l = local
z = other type of government publication
o = government publication -- level undetermined
u = unknown if item is government publication

| = no attempt to code

29 Conference publication:
0 = not a conference publication
1 = conference publication
| = no attempt to code

30 Festschrift:
0 = not a festschrift
1 = festschrift
| = no attempt to code

31 Index:
0 = no index
1 = index present
| = no attempt to code

32 Undefined (since 1990) (Earlier records may contain the values 0 or 1)
= Undefined
| = no attempt to code

33 Literary form:
0 = not fiction (not further specified)
1 = fiction (not further specified)
c = comic strips
d = dramas
e = essays
f = novels
h = humor, satires, etc.
i = letters
j = short stories
m = mixed forms
p = poetry
s = speeches
u = unknown
| = no attempt to code

34 Biography:
= no biographical material
a = autobiography
b = individual biography
c = collective biography
d = contains biographical information
| = no attempt to code

35-37 Language:
A three-letter code. For example: eng fre ger spa rus ita

(For a full list of codes used in these positions, see the *MARC Code List for Languages*.)

38 Modified record:
= not modified
x = missing characters (because of characters unavailable in MARC character set)
s = shortened
d = "dashed-on" information omitted
r = completely romanized/printed cards in script
o = completely romanized/printed cards romanized
| = no attempt to code

39 Cataloging source:
= national bibliographic agency
c = cooperative cataloging program
d = other sources
u = unknown
| = no attempt to code

[Back to Top of Page]

[Back to Table of Contents] -- [Continue to Part 11]



Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0150

Appendix Rellermeyer 01

```
LEADER 00000cam a2200553Ia 4500
001    IEEE827775858
006    m     o  d
007    cr cn|---|||||
008    120905s2012    njua    ob    100 0 eng d
020    9781467322645
020    1467322644
020    9781467322652
020    1467322652
040    W2U|beng|epn|cW2U|dOCLCO|dIEEXO|dOCLCF|dOCLCO|dOCLCQ|dOCL
       |dOCLCO|dOCLCQ|dIAS|dLND|dCEF|dOCLCQ|dLVT|dOCLCO
049    RVEU
050  4 QA76.5
099    ELECTRONIC|aRESOURCES
111 2  International Conference on Dependable Systems and
       Networks|n(42nd :|d2012 :|cBoston, Mass.).|eWorkshops
245 10 2012 IEEE/IFIP 42nd International Conference on Dependable
       Systems and Networks Workshops (DSN-W) :|bJune 25-28, 2012
       /|csponsored by IFIP WG 10.4 on Dependable Computing and
       Fault Tolerance, IEEE CS Technical Committee on Dependable
       Computing and Fault Tolerance, IEEE Computer Society
246 3  DSN-W 2012
246 1  |iAvailable from some providers with title:|aDependable
       Systems and Networks Workshops (DSN-W), 2012 IEEE/IFIP
       42nd International Conference on
246 30 Forty-second International Conference on Dependable
       Systems and Networks Workshops
246 30 42nd International Conference on Dependable Systems and
       Networks Workshops
246 30 IEEE/IFIP 42nd International Conference on Dependable
       Systems and Networks Workshops
260    [Piscataway, NJ] :|bIEEE Computer Society,|c©2012
300    1 online resource
336    text|btxt|2rdacontent
337    computer|bc|2rdamedia
338    online resource|bcr|2rdacarrier
490 0  IEEE Computer Society Conference Proceedings
500    Title from conference web page, viewed September 5, 2012
504    Includes bibliographical references
506    Access is limited to RIT students, faculty and staff
538    MODE OF ACCESS: World Wide Web
590    Subsequent record output
650  0 Application software|vCongresses
650  0 Fault-tolerant computing|vCongresses
655  7 Electronic books.|2local
710 2  IFIP Working Group 10.4 on Dependable Computing and Fault
       Tolerance
710 2  IEEE Computer Society.|bFault-Tolerant Computing Technical
       Committee
856 40 |zIEEE Electronic Books|uhttps://ezproxy.rit.edu/
       login?url=https://ieeexplore.ieee.org/servlet/
       opac?punumber=6255871
```

RIT Libraries

Hours | Chat | My Library Account

Start Over | Save Records | MARC Display | Another Search

Record number | b4043797 | Entire Catalog | Search

Limit search to available items





Permanent Link

Author International Conference on Dependable Systems and Networks (42nd : 2012 : Boston, Mass.). Workshops

Title 2012 IEEE/IFIP 42nd International Conference on Dependable Systems and Networks Workshops (DSN-W) : June 25-28, 2012 / sponsored by IFIP WG 10.4 on Dependable Computing and Fault Tolerance, IEEE CS Technical Committee on Dependable Computing and Fault Tolerance, IEEE Computer Society

Publisher [Piscataway, NJ] : IEEE Computer Society, ©2012

Available electronically:

IEEE Electronic Books

ISSN/ISBN 9781467322645
1467322644
9781467322652
1467322652

Descript 1 online resource

Content text

Media computer

Carrier online resource

Call No. QA76.5

Series IEEE Computer Society Conference Proceedings

Notes Title from conference web page, viewed September 5, 2012
Includes bibliographical references
Access is limited to RIT students, faculty and staff
MODE OF ACCESS: World Wide Web
Subsequent record output

Subject Application software -- Congresses
Fault-tolerant computing -- Congresses
Electronic books. local

Alt Author IFIP Working Group 10.4 on Dependable Computing and Fault Tolerance
IEEE Computer Society. Fault-Tolerant Computing Technical Committee

Alt Title Forty-second International Conference on Dependable Systems and Networks Workshops
42nd International Conference on Dependable Systems and Networks Workshops
IEEE/IFIP 42nd International Conference on Dependable Systems and Networks Workshops
DSN-W 2012
Available from some providers with title: Dependable Systems and Networks Workshops (DSN-W), 2012 IEEE/IFIP 42nd International Conference on

Contact Us | InterLibrary Loan

Appendix Rellermeyer 02

Dependability as a Cloud Service - A Modular Approach

Jan S. Rellermeyer[1]
[1]*IBM Research,*
Austin Research Lab,
Austin, TX 78758
rellermeyer@us.ibm.com

Saurabh Bagchi[1,2]
[2]*Purdue University,*
School of Electrical and Computer Engineering,
West Lafayette, IN 47907
sbagchi@purdue.edu, sbagchi@us.ibm.com

***Abstract*—Failures of services on cloud platforms are only to be expected. To deal with such failures, one is naturally inclined to use the traditional measure of replication. However, replication of services on distributed cloud platforms poses several challenges that are not well met by today's Java middleware systems. These challenges are the need to isolate state in the application components so that easy migration and recovery are possible and the requirement for client transparency when dealing with different replicated service instances. For example, Java Enterprise Edition (JEE) makes it difficult to have transparent replication of services due to the above two reasons plus the fine-grained nature of interactions between its components (the Enterprise Java Beans). In this paper, we show parts of the design of OSGi, a specification defining a dynamic component system in Java, that make it suitable for the above task. We then propose two extensions to OSGi which will allow exposing and exporting application component state and transparent invocation of service instances. These two together can enable easy replication and recovery from failures in cloud environments. We show through experiments that our prototype can migrate a failed service quickly enough to a new machine so that a client experiences only a moderate increase in service invocation time during system recovery.**

I. Introduction

A decision to move away from privately owned computer resources and towards cloud computing is affected by a variety of factors, dependability being one of them. Dependability is usually defined through its attributes of availability, reliability, safety, integrity, and maintainability [1]. In a complex system, the overall dependability of the system depends on the corresponding attributes of its components. What changes in cloud computing is that one important component—the platform—is no longer under one's own control. Depending on the type of the cloud service, the platform may mean one of several things; at least it includes the physical machine but may also include higher layers like a complete runtime system. This arguably outsources part of the maintenance problem while introducing new challenges. Most importantly, the underlying physical resources (i.e., the machines and the network) are shared between multiple customers and applications and cannot be actively managed. This is often perceived to be a threat to the overall dependability [2], [3], [4].

Typical ways of mitigating these threats in a cloud computing environment revolve around introducing redundancy into the system, especially for increased availability and reliability. Since adverse incidents can always render a single instance of a functional unit unusable, the hope is that by creating multiple copies of the component a sufficient number of copies always survive to deliver the service without degradation. This idea is neither new nor specific to cloud computing and has been successfully applied many many times in earlier distributed systems such as clusters or grids [5], [6]. In cloud computing, however, the effectiveness of redundancy benefits from two particular factors. First, the elasticity of the resource fabric makes redundancy more agile and more affordable. This means that it is possible to acquire additional resources for increasing the degree of redundancy in response to perceived threats much more easily than it can be done in traditional hosted environments. Second, the resource fabric is typically globally distributed, thereby allowing intelligent placement to guard against high-impact outages that would be much harder to circumvent in a hosted environment. By spanning multiple geographic zones, a system can survive even an outage of an entire data center that could be caused by a natural disaster or a far-reaching power outage.

Unfortunately, not every component of a complex system can easily profit from redundancy. Components that have state in them cannot be easily split up for purposes of replicating the parts to different replication degrees and executed on different machines. There have been prior attempts to address this by structuring web services such that most of the components are stateless and the few stateful components expose their state in an orderly manner. Thereby, it can be handled through special, and more expensive, mechanisms such as database replication. However, we discern that this kind of structuring of applications is becoming harder to do. Web services are becoming contextualized by various factors including the user, the client environment, the results of prior requests, and interactions with coordinating and collaborating web services. For example, with the advent of social networks, the services are contextualized by the user profiles and for a rich interactive user experience, the

978-1-4673-2266-9/12/$31.00 ©2012 IEEE

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0155

state (in this case, the profile) is driving the majority of the functionality. *Therefore, our position is that existing mechanisms for structuring applications for purposes of reliability will no longer work with emerging web services, that are being hosted on the cloud.*

The effectiveness of elasticity depends to a high degree on the coupling of components. Systems with fine granular components tend to suffer from a high degree of entanglement, hence limiting the flexibility of selectively replicating components since all tightly-coupled components usually have to reside on the same host. Consider for example, Enterprise Java Beans running on a JEE middleware are typically structured to have final granularity and with frequent invocations between them, both of which highlight the challenges of replication for cloud environments.

In this paper, we discuss how a dependability service can be built on the OSGi framework. OSGi is a set of specifications [7] that defines a dynamic component system for Java. These specifications enable a development model where applications are *dynamically* composed of many different reusable components. The OSGi specifications enable components to hide their implementations from other components while communicating through services, which are objects that are specifically shared between components. We show why a module system for Java, like OSGi, makes our goal simpler to achieve and then show how the dependability service can be architected using features already provided by OSGi. We propose an extension to the OSGi standard that acts as a dependability service, which we call CLOUDDEP. The service provides the following dependability properties:

1) Re-deployment: Migrating the application to a new physical machine or a different data center when the underlying platform fails.
2) Clustering: Eliminating single points of failure in the system by replicating functional units.
3) Elasticity and Load-balancing: Distributing the load to a flexible pool of resources to avoid overloading single instances.

II. SOFTWARE MODULARITY AND DEPENDABILITY

Software modules are self-contained functional units of encapsulation. They contain data and functions, objects, or other entities of the programming languages used to implement them. Ideally, the correctness of a modular system depends only on the correctness of its modules and the correctness of a single module does not depend on other modules [8]. The requirement of being self-contained, however does not automatically imply full isolation. Many implementations of module technology allow modules to be *declaratively self-contained*, i.e., they not only rely on their own content but can additionally declare dependencies to other modules and have access to their accessible content. In such setups, the effects of errors occurring in a single module are still restricted to cause faults in this module or any module that has a declared dependency but not in unrelated modules.

This property has been successfully exploited to increase the fault-tolerance of systems. For instance, in the Actor model [9] the modular units are fully isolated and can only communicate through explicit message passing.

Micro Kernel Operating Systems [10] restrict themselves to implement only the minimally necessary functionality in the kernel and outsource the remaining responsibilities into separate user-space modules. Thereby, faults in one module cannot affect other modules or the privileged kernel.

Microreboots [11] describe a methodology for recovering a system by rebooting only a subset of affected components, in the concrete example Enterprise Java Beans (EJBs). The design approach is to keep application state carefully parceled away in separate state stores which are only accessed through well-defined interfaces. Thus, the application components can be recovered by rebooting them (since the rebooting is fast in the absence of state, it is called microrebooting), while the state store can persist across reboots. This approach shares some similarities with our proposed approach. However, in order to be micro-rebootable, the components need to be well-isolated and stateless except for externalized application state kept in specialized state stores. Such compartmentalization is non-trivial to do in the context of JEE because of the reasons mentioned earlier (tight coupling and fine granular decomposition).

III. OSGI

A. OSGi Background

OSGi is a set of standards for modularity on the Java virtual machine released and maintained by the OSGi Alliance. Originally, it was developed by several major vendors to solve an availability problem on embedded home gateway machines [12]. The intention was that multiple software components should be able to co-exist on the same hardware while each of the software packages needs to run continuously and over long periods of time, even in scenarios that require periodic maintenance updates to the software. To enable dynamic updates without stoppage, it aimed at overcoming the traditional entanglement of small-granular objects and formed larger modules that can be managed individually and composed to form applications. OSGi defines a unit of modularity as the *Bundle*, which is a deployable Java JAR file enriched with additional meta-data. Most importantly, this meta-data describes which packages are exported, i.e., accessible to other bundles, and which packages it needs to import from other bundles. Recently, OSGi has successfully entered the domain of enterprise software and most major Java application servers (e.g., IBM WebSphere, JBoss, Oracle WebLogic, Spring Dynamic Module Platform) are by now internally based on the OSGi technology.

The OSGi runtime environment, the *Framework*, provides the user with mechanisms for installing and controlling the life-cycle of bundles at runtime. Applications are typically composed out of many different bundles. These bundles can interact in a tightly-coupled way by depending on packages provided by other bundles. *Tight coupling* in this context means that the dependencies are required to resolve the bundle and failure to provide the dependencies renders the bundle inoperable. Another way of interaction is through loosely-coupled services. *Loose coupling* means that service consumers only rely on the service interfaces under which the service has been registered with the OSGi framework but not on any knowledge of implementation details. Bundles can consequently query the framework for available services and retrieve the service instance, which is an ordinary Java object in OSGi. The result of a query is either a reference, a set of references from which the client can pick the best matching service, or the information that no such service is available. Since the consumers only rely on the service interfaces but not on any concrete implementation, bundles can potentially operate even in the absence of certain services. OSGi applications are expected to actively deal with dynamism in the setup. For instance, a service can certainly fulfill its purpose in the absence of a log service but if it has logging support, e.g., for debugging, it should enable it whenever a log service becomes available and disable it when the service becomes unavailable.

The modularity in OSGi solves a major issue with creating elastic applications for the cloud. Cloud resources are inherently dynamic in their nature and undergo frequent changes either due to explicit management operations (adding and removing resources) or due to their volatility (sharing effects, failures). Therefore, we believe that monolithic software incapable of dynamic adaptation cannot effectively run in such an environment. In fact, the compositional approach of modular software—traditionally applied to a single runtime system—is key to building scalable and dependable systems across a varying set of machines in the cloud [13], [14]. OSGi particularly facilitates composition and localized behavior by separating tightly coupled and loosely-coupled interaction between modules into two different concepts: package dependencies and services, respectively. Thereby, when services are used as a design element, the resulting systems are less entangled and more flexible.

OSGi was originally designed for managing software on a single Java virtual machine. In prior work, we have shown that the OSGi model can be used as an application model for building distributed systems by turning the services into potential boundaries of distribution [15] and adding simple support for creating redundancy by clustering stateless services on multiple machines [16]. However, little attention has so far been paid to the problem of managing stateful services in distributed deployments like clusters or clouds.

B. How OSGi Configuration Admin Enables Restarts

A traditional OSGi framework (running on a single JVM) already supports restarts of the runtime systems as well as a form of partial restart for bundle updates. This means that whenever a bundle has been updated and the operator of the framework requests a refresh of the system, only those bundles immediately affected by the update (this is the bundle itself and all bundles relying on the updated bundle through tightly-coupled package dependencies) are restarted whereas the remainder of the system is unaffected. However, only the compositional state of the application is preserved across restarts. This is the information about which bundles have been installed when the framework was last running and in which life-cycle state each individual bundle has been in (installed, resolved, starting, active, stopping, uninstalled). Every bundle then gets started again but is itself responsible for restoring any internal state, which includes services that it has bound to.

The only platform support in traditional OSGi for externalizing bundle state is the Configuration Admin service, an optional service described in the OSGi Compendium Specifications [17]. The configuration admin is a service in the system capable of managing and persisting configuration data in a central location. Services that want either to be externally configurable or want to persist their configuration data across restarts can register as a *Managed Service*. This expresses its intent to receive configuration updates and gives the system an interface through which it can push updates to the service. Every managed service is expected to identify itself though a persistent identifier (PID) and announce this PID with its initial registration. The config admin uses the PID of a service as a primary key for associating configuration data to services. Configuration events and changes are signaled to the managed service asynchronously, regardless of whether they are used by the service itself (as illustrated in Figure 1) or an external bundle.

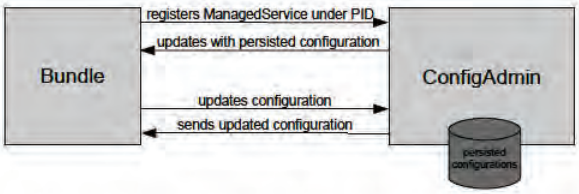


Figure 1. Interaction between Bundle and Configuration Admin

In the context of cloud computing, the most interesting and relevant property of the config admin is the concept of a service identity. When migrating a service to a new machine, it has to be assured that the newly created service instance can fully take over the role of the previous instance. Therefore, stateful services implicitly build up their own identity based on their internal state and their

history. Migration and failover can be considered a "micro-reboot" of the service on a different machine. If all state could be stored as configuration data then a distributed version of the configuration admin would turn managed services into migratable services just as the ConfigAdmin allows stateful services to become restartable. In practice, the storage capacity of the configuration admin and its unclear consistency model are prohibitive to this approach. For instance, a configuration update always fully overwrites any previous configuration so that when two writers are updating disjoint parts of the configuration concurrently the last writer wins and the update from the earlier writer is lost.

However, just as systems like Zookeeper [18] are used in cloud deployments to manage configuration, a cloud-scale version of an OSGi ConfigAdmin can be used to dynamically bind service instances to a richer instance of storage to externalize the service state, e.g., a database system or a key-value store. The challenge of building such a distributed ConfigAdmin for cloud deployments is to coordinate it with the distribution of services. It is not feasible to have a single scope for configurations across all nodes since this would prohibit multiple redundant copies of the same service. When a service needs to be migrated to a different machine, however, the configuration has to be migrated with the service.

C. How OSGi Remote Service Admin Enables Non-local Execution

The Remote Service Admin aims at integrating distribution into the OSGi platform and has been part of the OSGi Enterprise Specifications [19]. This service encapsulates the mechanism for importing remote services into the scope of a local OSGi framework and exporting local OSGi services through distribution providers. When a remote service is imported, a local proxy for the remote service is created and associated with the distribution provider. Some distribution providers like our R-OSGi [15] system are able to operate transparently, i.e., neither client nor service have to be rewritten to operate in a remote setup.

The API of the remote service admin is entirely imperative. Given a known URI or other identifier, the admin provisions the local proxy and handles binding to the remote service reachable through this address, as long as the service is reachable. Finding a service in the network and re-binding in the event of a failure is handled by different components. All policy decisions are encapsulated in a Topology Manager which, e.g., is able to detect clients that are querying the local framework for services and make use of service discovery protocols to find matching services. However, the specifications do not mandate any particular functionality of the topology manager but only specify the interfaces through which a possible topology manager can interact.

For a cloud setup with dependability requirements, the presence of an appropriate topology manager is paramount.

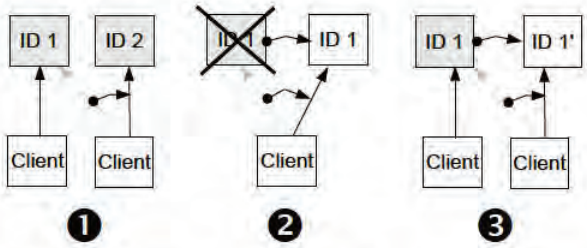


Figure 2. Different Use Cases for a Dependability Service. 1: Load balancing, 2: Failover, 3: Elastic Scalability.

In the following section, we describe our service CLOUDDEP that implements policies to ensure availability of services by acting as both a topology manager and a distributed configuration admin.

IV. TOWARDS A DEPENDABILITY SERVICE

The new service (that we call CLOUDDEP) is designed to turn OSGi services into elastic units of deployment which can be migrated and replicated across multiple cloud nodes to increase the availability and reliability of the administered service. All that is required from the administered service is that it persists all of its internal state either directly or indirectly through configurations and listens to configuration updates by implementing the Managed Service interface of the configuration admin specification. CLOUDDEP acts as a global entity in the system (which can be implemented by a centralized backend or a distributed architecture) and takes over the role of a local configuration admin and a local topology manager on each node of the cloud deployment. It also monitors the service invocations to detect failures. We illustrate the features of CLOUDDEP through three usage cases.

A. Load Balancing

We consider a simple setup with a set of identical (possibly stateful) remote service and a set of service clients as illustrated by case ❶ in Figure 2. Thanks to its facet of being a topology manager, CLOUDDEP has knowledge of the remote service relationship and was involved in the setup of the binding. It had detected the local service requests of the client bundles (e.g., through service lookup hooks as described in the remote service admin specification) and has instructed the distribution provider to create the binding to one of the known remote service instances. Furthermore, it has, either due to cooperation of the distribution provider or by wrapping its created service proxy, instrumented the local proxy to gather information about the frequencies of service invocation and response times. In the event that one remote service instance becomes overloaded, detectable by an increase of invocations and an increase of the response time, CLOUDDEP can re-bind a client to a different service

instance, thereby re-distributing the load among the service instances.

B. Failover

We now consider a setup with one or more clients and a single remote service. In the event of a service failure (e.g., an uncaught runtime exception or the node hosting the remote service becoming unavailable) CLOUDDEP has captured the identity of the faulty service through its configuration admin facet. It can thereby instantiate a new service by starting the corresponding bundle on a new node and inject the identity from the failed service into the new instance, as shown in case ❷. This process is akin to what a microreboot would do on a single node but here applied to a distributed setup. By applying the same rebinding as done for load balancing, the existing client will be forced to now interact with the new service instance. If the latency of service migration is an issue, CLOUDDEP can be configured to keep hot-spare instances of the service. These are running but unconfigured instances that have not yet received an identity. In the event of a service failure, CLOUDDEP can directly inject the identity into a hot-spare instance and create a more seamless failover.

C. Elastic Scalability

The third example combines the load balancing and the failover scenarios to enable elastic scalability of a remote service. In the event of a service over-utilization, CLOUDDEP can start a new instance of the service and bind some of the clients to the new service (see ❸ in Figure 2). A new identity is forked by CLOUDDEP from an existing configuration and can then develop independently from the original sibling. Scaling back to a smaller number of service instances can be achieved by applying the failover strategy: moving all clients of a service to a different existing instance and then dropping the service instance.

V. PROTOTYPE AND EXPERIMENTAL EVALUATION

We have implemented a prototype of CLOUDDEP that replicates configuration and manages failover for OSGi services in a cloud setup. The prototype uses R-OSGi as a distribution provider and cooperates with the protocol and the proxy handler to detect failures or service outages. The detection of failure is done through either a simple timeout or an exception caused by the termination of a connection. Since we induce failures in the experiments through terminating the connection, the latter detection is triggered. CLOUDDEP can furthermore either proactively start idle and unconfigured redundant instances of the service on a backup machine or reactively start a new instance of the service on demand. In either case, the identity of the failed service is injected into the new instance and the existing binding between the client and the no longer available service is transparently altered to point to the new instance. The prototype blocks the failed service request and replays the request to the new service instance as soon as it becomes available. We therefore assume a fail-stop model in our setup.

	Microreboot	Migration
avg. inv. time (T)	$7.23\mu s \pm 1.06\mu s$	$1.22ms \pm 0.15ms$
avg. T during failover	$266.29\mu s \pm 34.90\mu s$	$4.01ms \pm 0.66ms$
relative overhead	3581%	229%

Table I
SERVICE INVOCATION LATENCIES OF THE TREND SERVICE

For the experiment, we use a trend service that updates itself by periodically polling the most frequently used #hashtags from Twitter. The service is clustered by geographic regions, each instance of the service is configured to handle a specific region that is set through its configuration. Since the service only handles the ten most relevant results, it can afford to use the Configuration Admin interface to persist its internal state and does not rely on, e.g., an external blob storage. In a realistic setup, the service would process the results and, for instance, enrich the data with context links retrieved from Google Search or Wikipedia.

We use a single instance of the service and a single client that polls the service every second to receive the latest trends. After one minute, we take the service down and let CLOUDDEP handle the failure based on a strategy that we have pre-configured.

In the first experiment, we have both the client and the service running in the same Java Virtual Machine and on the same OSGi framework. Our failure recovery strategy resembles a microreboot of the service, i.e., we stop the running service and restart it. In the second experiment, we run client and service in different cloud instances and on different Java Virtual Machines and additionally keep a hot-spare VM for the failover. When the original service fails, we instantiate a new service instance on the hot-spare that becomes enabled when CLOUDDEP injects the configuration (i.e., geographic location and previously determined trends) into the instance.

Table I summarizes the measured average service invocation time during normal operation and during the failover. In the case of the microreboot, the invocation time during failover increases by a factor of 35 during failover. The reason for this high increase is that in OSGi clients get a direct reference to the service object and thereby a regular service invocation does not differ from any other method call in terms of latency. Enterprise Java Beans, in contrast, are called through reflection. Furthermore, a bundle restart is potentially more expensive than a microreboot of a single Enterprise Bean. However, the overall latency during failover is still in the range of microseconds, which is very likely acceptable for the large majority of applications.

In the distributed case the baseline of an average regular

service invocation is much higher and is in our setup in the order of milliseconds. However, the failover increases the invocation time by only a factor of three. In practice, this factor will vary with the amount of configuration data that needs to be migrated to restore the service identity and the network used but more complex services are likely to also have higher baseline invocation times. In both the local and the distributed case, only a single service invocation is temporarily blocked and experiences a higher latency. Further invocations continue to operate at normal latency.

The results illustrate the viability of our approach for a large class of applications. Microreboots are an option for recovery from failures, especially when the service is already accessed from a different host. In this case, the latency of the microreboot is within the noise of a remote service invocation. When entire nodes fail or become unreachable, migration of services to different machines are also feasible and do not affect the client significantly. We have, however, assumed that the cloud instance for failover is already instantiated. In our setup, we used an Eucalyptus cluster that requires in average 85 seconds to boot up a new instance to the point where we can interact with it. This latency is unlikely to be acceptable for cloud services but is a common problem of IaaS solutions.

VI. Conclusion

We expect dependability to become an increasing problem for cloud services. Our premise is that traditional ways of structuring such systems provide insufficient flexibility and cannot deal with failures or elasticity requirements as they occur in cloud setups. We have therefore proposed a modular, loosely-coupled approach for building cloud services and have illustrated the concrete challenges using the example of OSGi, a widely used standard for modularity for the Java language. The result of this effort is CloudDep, a dependability service for OSGi that takes care of countering service failures by migrating the identity of the service to a new machine. The experimental results of our prototype implementation show that migration to an existing node is feasible and the impact on the service invocation time is acceptable for most applications.

References

[1] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, 2004.

[2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, Apr. 2010.

[3] "Cloud Computing Incidents Database." [Online]. Available: http://wiki.cloudcommunity.org/wiki/CloudComputing:Incidents_Database

[4] K. Joshi, G. Bunker, F. Jahanian, A. van Moorsel, and J. Weinman, "Dependability in the cloud: Challenges and opportunities," in *Dependable Systems Networks, 2009. DSN '09. IEEE/IFIP International Conference on*, 29 2009-july 2 2009, pp. 103 –104.

[5] H. Stockinger, A. Samar, K. Holtman, B. Allcock, I. Foster, and B. Tierney, "File and object replication in data grids," *Cluster Computing*, vol. 5, no. 3, pp. 305–314, 2002.

[6] J. Abawajy, "Fault-tolerant scheduling policy for grid computing systems," in *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*. IEEE, 2004, pp. 238–244.

[7] OSGi Alliance, *OSGi Service Platform, Core Specification Release 4, Version 4.2*, 2009.

[8] B. Meyer, *Object-Oriented Software Construction*, 1st ed. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1988.

[9] C. Hewitt, P. Bishop, and R. Steiger, "A universal modular ACTOR formalism for artificial intelligence," in *Proceedings of the 3rd international joint conference on Artificial intelligence*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1973, pp. 235–245.

[10] J. Liedtke, "On micro-kernel construction," in *Proceedings of the fifteenth ACM symposium on Operating systems principles*, ser. SOSP '95. New York, NY, USA: ACM, 1995, pp. 237–250.

[11] G. Candea, S. Kawamoto, Y. Fujiki, G. Friedman, and A. Fox, "Microreboot - a technique for cheap recovery," in *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, ser. OSDI'04, 2004.

[12] P. Kriens, "How osgi changed my life," *ACM Queue*, vol. 6, no. 1, Jan. 2008.

[13] C. Matthews and Y. Coady, "Virtualized recomposition: Cloudy or clear?" *Software Engineering Challenges of Cloud Computing, ICSE Workshop on*, vol. 0, pp. 38–43, 2009.

[14] J. S. Rellermeyer, M. Duller, and G. Alonso, "Engineering the cloud from software modules," in *ICSE-Cloud 09: Proceedings of the Workshop on Software Engineering Challenges in Cloud Computing (in conjunction with ICSE 2009)*, Vancouver, Canada, 2009.

[15] J. S. Rellermeyer, G. Alonso, and T. Roscoe, "R-osgi: Distributed applications through software modularization," in *Middleware 07: 8th ACM/IFIP/USENIX International Middleware Conference*, Newport Beach, CA, USA, 2007.

[16] J. S. Rellermeyer, G. Alonso, and T. Roscoe, "Building, deploying, and monitoring distributed applications with eclipse and r-osgi," in *ETX 07: Fifth Eclipse Technology eXchange (ETX) Workshop (in conjunction with OOPSLA 2007)*, Montreal, Canada, 2007.

[17] OSGi Alliance, *OSGi Service Platform, Service Compendium Release 4, Version 4.2*, 2009.

[18] Apache Foundation, "Apache zookeeper," http://zookeeper.apache.org, 2008.

[19] OSGi Alliance, *OSGi Service Platform Enterprise Specifications, Release 4, Version 4.2*, 2010.

Appendix Rellermeyer 03

IEEE.org IEEE *Xplore* IEEE SA IEEE Spectrum More Sites Subscribe Donate Cart Welcome Charles Dougherty Sign Out

Browse My Settings Help Institutional Sign In

Institutional Sign In

All

ADVANCED SEARCH

Conferences > IEEE/IFIP International Confe...

Dependability as a cloud service - a modular approach

Publisher: IEEE Cite This PDF

Jan S. Rellermeyer ; Saurabh Bagchi **All Authors**

4 Cites in Papers

281 Full Text Views

Alerts

Manage Content Alerts

Add to Citation Alerts

Abstract

Document Sections

I. Introduction

II. Software Modularity and Dependability

III. OSGi

IV. Towards a Dependability Service

V. Prototype And Experimental Evaluation

Show Full Outline

Authors

Figures

References

Citations

Keywords

Metrics

More Like This

Downl PDF

Abstract:
Failures of services on cloud platforms are only to be expected. To deal with such failures, one is naturally inclined to use the traditional measure of replication. Howe... **View more**

Metadata

Abstract:
Failures of services on cloud platforms are only to be expected. To deal with such failures, one is naturally inclined to use the traditional measure of replication. However, replication of services on distributed cloud platforms poses several challenges that are not well met by today's Java middleware systems. These challenges are the need to isolate state in the application components so that easy migration and recovery are possible and the requirement for client transparency when dealing with different replicated service instances. For example, Java Enterprise Edition (JEE) makes it difficult to have transparent replication of services due to the above two reasons plus the fine-grained nature of interactions between its components (the Enterprise Java Beans). In this paper, we show parts of the design of OSGi, a specification defining a dynamic component system in Java, that make it suitable for the above task. We then propose two extensions to OSGi which will allow exposing and exporting application component state and transparent invocation of service instances. These two together can enable easy replication and recovery from failures in cloud environments. We show through experiments that our prototype can migrate a failed service quickly enough to a new machine so that a client experiences only a moderate increase in service invocation time during system recovery.

Published in: IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)

Date of Conference: 25-28 June 2012

DOI: 10.1109/DSNW.2012.6264688

Date Added to IEEE *Xplore*: 13 August 2012

Publisher: IEEE

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0162

Conference Location: Boston, MA, USA

ISBN Information:

Electronic ISBN:978-1-4673-2266-9
Print ISBN:978-1-4673-2264-5
Online ISBN:978-1-4673-2265-2

ISSN Information:

Print ISSN: 2325-6648
Electronic ISSN: 2325-6664

Contents

SECTION I.
Introduction

A decision to move away from privately owned computer resources and towards cloud computing is affected by a variety of factors, dependability being one of them. Dependability is usually defined through its attributes of availability, reliability, safety, integrity, and maintainability [1]. In a complex system, the overall dependability of the system depends on the corresponding attributes of its components. What changes in cloud computing is that one important component—the platform—is no longer under one's own control. Depending on the type of the cloud service, the platform may mean one of several things; at least it includes the physical machine but may also include higher layers like a complete runtime system. This arguably outsources part of the maintenance problem while introducing new challenges. Most importantly, the underlying physical resources (i.e., the machines and the network) are shared between multiple customers and applications and cannot be actively managed. This is often perceived to be a threat to the overall dependability [2], [3], [4].

Typical ways of mitigating these threats in a cloud computing environment revolve around introducing redundancy into the system, especially for increased availability and reliability. Since adverse incidents can always render a single instance of a functional unit unusable, the hope is that by creating multiple copies of the component a sufficient number of copies always survive to deliver the service without degradation. This idea is neither new nor specific to cloud computing and has been successfully applied many many times in earlier distributed systems such as clusters or grids [5], [6]. In cloud computing, however, the effectiveness of redundancy benefits from two particular factors. First, the elasticity of the resource fabric makes redundancy more agile and more affordable. This means that it is possible to acquire additional resources for increasing the degree of redundancy in response to perceived threats much more easily than it can be done in traditional hosted environments. Second, the resource fabric is typically globally distributed, thereby allowing intelligent placement to guard against high-impact outages that would be much harder to circumvent in a hosted environment. By spanning multiple geographic zones, a system can survive even an outage of an entire data center that could be caused by a natural disaster or a far-reaching power outage.

Unfortunately, not every component of a complex system can easily profit from redundancy. Components that have state in them cannot be easily split up for purposes of replicating the parts to different replication degrees and executed on different machines. There have been prior attempts to address this by structuring web services such that most of the components are stateless and the few stateful components expose their state in an orderly manner. Thereby, it can be handled through special, and more expensive, mechanisms such as database replication. However, we discern that this kind of structuring of applications is becoming harder to do. Web services are becoming contextualized by various factors including the user, the client environment, the results of prior requests, and interactions with coordinating and collaborating web services. For example, with the advent of social networks, the services are contextualized by the user profiles and for a rich interactive user experience, the state (in this case, the profile) is driving the majority of the functionality. *Therefore, our position is that existing mechanisms for structuring applications for purposes of reliability will no longer work with emerging web services, that are being hosted on the cloud.*

The effectiveness of elasticity depends to a high degree on the coupling of components. Systems with fine granular components tend to suffer from a high degree of entangle-ment, hence limiting the flexibility of

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0163

selectively replicating components since all tightly-coupled components usually have to reside on the same host. Consider for example, Enterprise Java Beans running on a JEE middleware are typically structured to have final granularity and with frequent invocations between them, both of which highlight the challenges of replication for cloud environments.

In this paper, we discuss how a dependability service can be built on the OSGi framework. OSGi is a set of specifications [7] that defines a dynamic component system for Java. These specifications enable a development model where applications are *dynamically* composed of many different reusable components. The OSGi specifications enable components to hide their implementations from other components while communicating through services, which are objects that are specifically shared between components. We show why a module system for Java, like OSGi, makes our goal simpler to achieve and then show how the dependability service can be architected using features already provided by OSGi. We propose an extension to the OSGi standard that acts as a dependability service, which we call Cloud dep. The service provides the following dependability properties:

1) Re-deployment: Migrating the application to a new physical machine or a different data center when the underlying platform fails.

2) Clustering: Eliminating single points of failure in the system by replicating functional units.

3) Elasticity and Load-balancing: Distributing the load to a flexible pool of resources to avoid overloading single instances.

SECTION II.
Software Modularity and Dependability

Software modules are self-contained functional units of encapsulation. They contain data and functions, objects, or other entities of the programming languages used to implement them. Ideally, the correctness of a modular system depends only on the correctness of its modules and the correctness of a single module does not depend on other modules [8]. The requirement of being self-contained, however does not automatically imply full isolation. Many implementations of module technology allow modules to be *declaratively self-contained*, i.e., they not only rely on their own content but can additionally declare dependencies to other modules and have access to their accessible content. In such setups, the effects of errors occurring in a single module are still restricted to cause faults in this module or any module that has a declared dependency but not in unrelated modules.

This property has been successfully exploited to increase the fault-tolerance of systems. For instance, in the Actor model [9] the modular units are fully isolated and can only communicate through explicit message passing.

Micro Kernel Operating Systems [10] restrict themselves to implement only the minimally necessary functionality in the kernel and outsource the remaining responsibilities into separate user-space modules. Thereby, faults in one module cannot affect other modules or the privileged kernel.

Microreboots [11] describe a methodology for recovering a system by rebooting only a subset of affected components, in the concrete example Enterprise Java Beans (EJBs). The design approach is to keep application state carefully parceled away in separate state stores which are only accessed through well-defined interfaces. Thus, the application components can be recovered by rebooting them (since the rebooting is fast in the absence of state, it is called microre-booting), while the state store can persist across reboots. This approach shares some similarities with our proposed approach. However, in order to be micro-rebootable, the components need to be well-isolated and stateless except for externalized application state kept in specialized state stores. Such compartmentalization is non-trivial to do in the context of JEE because of the reasons mentioned earlier (tight coupling and fine granular decomposition).

SECTION III.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0164

OSGi

A. OSGi Background

OSGi is a set of standards for modularity on the Java virtual machine released and maintained by the OSGi Al-liance. Originally, it was developed by several major ven-dors to solve an availability problem on embedded home gateway machines [12]. The intention was that multiple software components should be able to co-exist on the same hardware while each of the software packages needs to run continuously and over long periods of time, even in scenarios that require periodic maintenance updates to the software. To enable dynamic updates without stoppage, it aimed at overcoming the traditional entanglement of small-granular objects and formed larger modules that can be managed individually and composed to form applications. OSGi defines a unit of modularity as the *Bundle*, which is a deployable Java JAR file enriched with additional meta-data. Most importantly, this meta-data describes which packages are exported, i.e., accessible to other bundles, and which packages it needs to import from other bundles. Recently, OSGi has successfully entered the domain of enterprise software and most major Java application servers (e.g., IBM WebSphere, JBoss, Oracle WebLogic, Spring Dynamic Module Platform) are by now internally based on the OSGi technology.

The OSGi runtime environment, the *Framework*, provides the user with mechanisms for installing and controlling the life-cycle of bundles at runtime. Applications are typically composed out of many different bundles. These bundles can interact in a tightly-coupled way by depending on packages provided by other bundles. *Tight coupling* in this context means that the dependencies are required to resolve the bundle and failure to provide the dependencies renders the bundle inoperable. Another way of interaction is through loosely-coupled services. *Loose coupling* means that service consumers only rely on the service interfaces under which the service has been registered with the OSGi framework but not on any knowledge of implementation details. Bundles can consequently query the framework for available services and retrieve the service instance, which is an ordinary Java object in OSGi. The result of a query is either a reference, a set of references from which the client can pick the best matching service, or the information that no such service is available. Since the consumers only rely on the service interfaces but not on any concrete implementation, bundles can potentially operate even in the absence of certain services. OSGi applications are expected to actively deal with dynamism in the setup. For instance, a service can certainly fulfill its purpose in the absence of a log service but if it has logging support, e.g., for debugging, it should enable it whenever a log service becomes available and disable it when the service becomes unavailable.

The modularity in OSGi solves a major issue with creating elastic applications for the cloud. Cloud resources are inher-ently dynamic in their nature and undergo frequent changes either due to explicit management operations (adding and re-moving resources) or due to their volatility (sharing effects, failures). Therefore, we believe that monolithic software incapable of dynamic adaptation cannot effectively run in such an environment. In fact, the compositional approach of modular software-traditionally applied to a single runtime system-is key to building scalable and dependable systems across a varying set of machines in the cloud [13], [14]. OSGi particularly facilitates composition and localized behavior by separating tightly coupled and loosely-coupled interaction between modules into two different concepts: package dependencies and services, respectively. Thereby, when services are used as a design element, the resulting systems are less entangled and more flexible.

OSGi was originally designed for managing software on a single Java virtual machine. In prior work, we have shown that the OSGi model can be used as an application model for building distributed systems by turning the services into potential boundaries of distribution [15] and adding simple support for creating redundancy by clustering stateless services on multiple machines [16]. However, little attention has so far been paid to the problem of managing stateful services in distributed deployments like clusters or clouds.

B. How OSGi Configuration Admin Enables Restarts

A traditional OSGi framework (running on a single JVM) already supports restarts of the runtime systems as well as a form of partial restart for bundle updates. This means that whenever a bundle has been updated and the operator of the framework requests a refresh of the system, only those bundles immediately affected by the update (this is the bundle itself and all bundles relying on the updated bundle through tightly-coupled package dependencies) are restarted whereas the remainder of the system is unaffected. However, only the compositional state of the application is preserved across restarts. This is the information about which bundles have been installed when the framework was last running and in which life-cycle state each individual bundle has been in (installed, resolved, starting, active, stopping, uninstalled). Every bundle then gets started again but is itself responsible for restoring any internal state, which includes services that it has bound to.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0165

The only platform support in traditional OSGi for exter-nalizing bundle state is the Configuration Admin service, an optional service described in the OSGi Compendium Specifications [17]. The configuration admin is a service in the system capable of managing and persisting configuration data in a central location. Services that want either to be externally configurable or want to persist their configuration data across restarts can register as a *Managed Service*. This expresses its intent to receive configuration updates and gives the system an interface through which it can push updates to the service. Every managed service is expected to identify itself though a persistent identifier (PID) and announce this PID with its initial registration. The config admin uses the PID of a service as a primary key for associ-ating configuration data to services. Configuration events and changes are signaled to the managed service asynchronously, regardless of whether they are used by the service itself (as illustrated in Figure 1) or an external bundle.

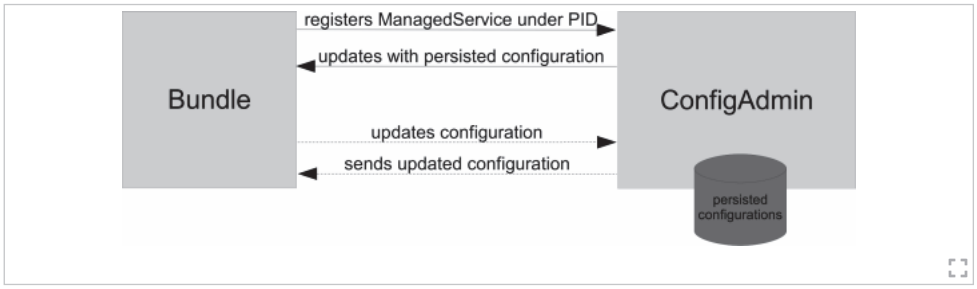


Figure 1. Interaction between Bundle and Configuration Admin

In the context of cloud computing, the most interesting and relevant property of the config admin is the concept of a service identity. When migrating a service to a new machine, it has to be assured that the newly created service instance can fully take over the role of the previous instance. Therefore, stateful services implicitly build up their own identity based on their internal state and their history. Migration and failover can be considered a “micro-reboot” of the service on a different machine. If all state could be stored as configuration data then a distributed version of the configuration admin would turn managed services into migratable services just as the ConfigAdmin allows stateful services to become restartable. In practice, the storage capacity of the configuration admin and its unclear consistency model are prohibitive to this approach. For instance, a configuration update always fully overwrites any previous configuration so that when two writers are updating disjoint parts of the configuration concurrently the last writer wins and the update from the earlier writer is lost.

However, just as systems like Zookeeper [18] are used in cloud deployments to manage configuration, a cloud-scale version of an OSGi ConfigAdmin can be used to dynamically bind service instances to a richer instance of storage to externalize the service state, e.g., a database system or a key-value store. The challenge of building such a distributed ConfigAdmin for cloud deployments is to coordinate it with the distribution of services. It is not feasible to have a single scope for configurations across all nodes since this would prohibit multiple redundant copies of the same service. When a service needs to be migrated to a different machine, however, the configuration has to be migrated with the service.

C. How OSGi Remote Service Admin Enables Non-local Execution

The Remote Service Admin aims at integrating distribution into the OSGi platform and has been part of the OSGi Enterprise Specifications [19]. This service encapsulates the mechanism for importing remote services into the scope of a local OSGi framework and exporting local OSGi services through distribution providers. When a remote service is imported, a local proxy for the remote service is created and associated with the distribution provider. Some distribution providers like our R-OSGi [15] system are able to operate transparently, i.e., neither client nor service have to be rewritten to operate in a remote setup.

The API of the remote service admin is entirely imper-ative. Given a known URI or other identifier, the admin provisions the local proxy and handles binding to the remote service reachable through this address, as long as the service is reachable. Finding a service in the network and re-binding in the event of a failure is handled by different components. All policy decisions are encapsulated in a Topology Manager which, e.g., is able to detect clients that are querying the local framework for services and make use of service discovery protocols to find matching services. However, the specifications do not mandate any particular functionality of the topology manager but only specify the interfaces through which a possible topology manager can interact.

For a cloud setup with dependability requirements, the presence of an appropriate topology manager is paramount. In the following section, we describe our service Clouddep that implements policies to ensure

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0166

availability of services by acting as both a topology manager and a distributed configuration admin.

SECTION IV.
Towards a Dependability Service

The new service (that we call CloudDep) is designed to turn OSGi services into elastic units of deployment which can be migrated and replicated across multiple cloud nodes to increase the availability and reliability of the administered service. All that is required from the administered service is that it persists all of its internal state either directly or indirectly through configurations and listens to configuration updates by implementing the Managed Service interface of the configuration admin specification. CloudDep acts as a global entity in the system (which can be implemented by a centralized backend or a distributed architecture) and takes over the role of a local configuration admin and a local topology manager on each node of the cloud deployment. It also monitors the service invocations to detect failures. We illustrate the features of CloudDep through three usage cases.

A. Load Balancing

We consider a simple setup with a set of identical (pos-sibly stateful) remote service and a set of service clients as illustrated by case ❶ in Figure 2. Thanks to its facet of being a topology manager, Clouddep has knowledge of the remote service relationship and was involved in the setup of the binding. It had detected the local service requests of the client bundles (e.g., through service lookup hooks as described in the remote service admin specification) and has instructed the distribution provider to create the binding to one of the known remote service instances. Furthermore, it has, either due to cooperation of the distribution provider or by wrapping its created service proxy, instrumented the local proxy to gather information about the frequencies of service invocation and response times. In the event that one remote service instance becomes overloaded, detectable by an increase of invocations and an increase of the response time, CloudDep can re-bind a client to a different service instance, thereby re-distributing the load among the service instances.

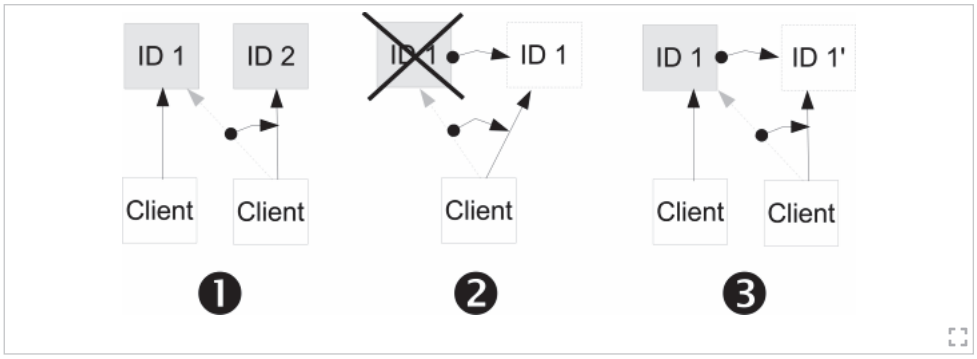


Figure 2. Different Use Cases for a Dependability Service. 1: Load balancing, 2: Failover, 3: Elastic Scalability.

B. Failover

We now consider a setup with one or more clients and a single remote service. In the event of a service failure (e.g., an uncaught runtime exception or the node hosting the remote service becoming unavailable) CloudDep has captured the identity of the faulty service through its configuration admin facet. It can thereby instantiate a new service by starting the corresponding bundle on a new node and inject the identity from the failed service into the new instance, as shown in case ❷. This process is akin to what a microreboot would do on a single node but here applied to a distributed setup. By applying the same rebinding as done for load balancing, the existing client will be forced to now interact with the new service instance. If the latency of service migration is an issue, CloudDep can be configured to keep hot-spare instances of the service. These are running but unconfigured instances that have not yet received an identity. In the event of a service failure, CloudDep can directly inject the identity into a hot-spare instance and create a more seamless failover.

C. Elastic Scalability

The third example combines the load balancing and the failover scenarios to enable elastic scalability of a remote service. In the event of a service over-utilization, CloudDep can start a new instance of the service and

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0167

bind some of the clients to the new service (see ❸ in Figure 2). A new identity is forked by CloudDep from an existing configuration and can then develop independently from the original sibling. Scaling back to a smaller number of service instances can be achieved by applying the failover strategy: moving all clients of a service to a different existing instance and then dropping the service instance.

SECTION V.
Prototype And Experimental Evaluation

We have implemented a prototype of CloudDep that replicates configuration and manages failover for OSGi services in a cloud setup. The prototype uses R-OSGi as a distribution provider and cooperates with the protocol and the proxy handler to detect failures or service outages. The detection of failure is done through either a simple timeout or an exception caused by the termination of a connection. Since we induce failures in the experiments through terminating the connection, the latter detection is triggered. CloudDep can furthermore either proactively start idle and unconfigured redundant instances of the service on a backup machine or reactively start a new instance of the service on demand. In either case, the identity of the failed service is injected into the new instance and the existing binding between the client and the no longer available service is transparently altered to point to the new instance. The prototype blocks the failed service request and replays the request to the new service instance as soon as it becomes available. We therefore assume a fail-stop model in our setup.

For the experiment, we use a trend service that updates itself by periodically polling the most frequently used #hash-tags from Twitter. The service is clustered by geographic regions, each instance of the service is configured to handle a specific region that is set through its configuration. Since the service only handles the ten most relevant results, it can afford to use the Configuration Admin interface to persist its internal state and does not rely on, e.g., an external blob storage. In a realistic setup, the service would process the results and, for instance, enrich the data with context links retrieved from Google Search or Wikipedia.

We use a single instance of the service and a single client that polls the service every second to receive the latest trends. After one minute, we take the service down and let CloudDep handle the failure based on a strategy that we have pre-configured.

In the first experiment, we have both the client and the service running in the same Java Virtual Machine and on the same OSGi framework. Our failure recovery strategy resembles a microreboot of the service, i.e., we stop the running service and restart it. In the second experiment, we run client and service in different cloud instances and on different Java Virtual Machines and additionally keep a hot-spare VM for the fail over. When the original service fails, we instantiate a new service instance on the hot-spare that becomes enabled when CloudDep injects the configuration (i.e., geographic location and previously determined trends) into the instance.

Table I summarizes the measured average service invocation time during normal operation and during the failover. In the case of the microreboot, the invocation time during failover increases by a factor of 35 during failover. The reason for this high increase is that in OSGi clients get a direct reference to the service object and thereby a regular service invocation does not differ from any other method call in terms of latency. Enterprise Java Beans, in contrast, are called through reflection. Furthermore, a bundle restart is potentially more expensive than a microreboot of a single Enterprise Bean. However, the overall latency during failover is still in the range of microseconds, which is very likely acceptable for the large majority of applications.

Table I Service Invocation Latencies of the Trend Service

	Microreboot	Migration
avg. inv. time (T)	$7.23\mu s \pm 1.06\mu s$	$1.22ms \pm 0.15ms$
avg. T during failover	$266.29\mu s \pm 34.90\mu s$	$4.01ms \pm 0.66ms$
relative overhead	3581%	229%

In the distributed case the baseline of an average regular service invocation is much higher and is in our setup in the order of milliseconds. However, the failover increases the invocation time by only a factor of three. In practice, this factor will vary with the amount of configuration data that needs to be migrated to restore the

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0168

service identity and the network used but more complex services are likely to also have higher baseline invocation times. In both the local and the distributed case, only a single service invocation is temporarily blocked and experiences a higher latency. Further invocations continue to operate at normal latency.

The results illustrate the viability of our approach for a large class of applications. Microreboots are an option for recovery from failures, especially when the service is already accessed from a different host. In this case, the latency of the microreboot is within the noise of a remote service invocation. When entire nodes fail or become unreachable, migration of services to different machines are also feasible and do not affect the client significantly. We have, however, assumed that the cloud instance for failover is already instantiated. In our setup, we used an Eucalyptus cluster that requires in average 85 seconds to boot up a new instance to the point where we can interact with it. This latency is unlikely to be acceptable for cloud services but is a common problem of IaaS solutions.

SECTION VI.
Conclusion

We expect dependability to become an increasing problem for cloud services. Our premise is that traditional ways of structuring such systems provide insufficient flexibility and cannot deal with failures or elasticity requirements as they occur in cloud setups. We have therefore proposed a modu-lar, loosely-coupled approach for building cloud services and have illustrated the concrete challenges using the example of OSGi, a widely used standard for modularity for the Java language. The result of this effort is CloudDep, a dependability service for OSGi that takes care of countering service failures by migrating the identity of the service to a new machine. The experimental results of our prototype implementation show that migration to an existing node is feasible and the impact on the service invocation time is acceptable for most applications.

Authors
Figures
References
Citations
Keywords
Metrics

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0169

System management to comply with SLA availability guarantees in cloud computing
4th IEEE International Conference on Cloud Computing Technology and Science Proceedings
Published: 2012

Enhancing Reliability and Availability through Redundancy in Vehicular Clouds
IEEE Transactions on Cloud Computing
Published: 2021

Show More

IEEE Personal Account

CHANGE USERNAME/PASSWORD

Purchase Details

PAYMENT OPTIONS

VIEW PURCHASED DOCUMENTS

Profile Information

COMMUNICATIONS PREFERENCES

PROFESSION AND EDUCATION

TECHNICAL INTERESTS

Need Help?

US & CANADA: +1 800 678 4333

WORLDWIDE: +1 732 981 0060

CONTACT & SUPPORT

Follow

About IEEE *Xplore* | Contact Us | Help | Accessibility | Terms of Use | Nondiscrimination Policy | IEEE Ethics Reporting | Sitemap | IEEE Privacy Policy

A public charity, IEEE is the world's largest technical professional organization dedicated to advancing technology for the benefit of humanity.

© Copyright 2025 IEEE - All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies.

IEEE Account

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0170

» Change Username/Password

» Update Address

Purchase Details

» Payment Options

» Order History

» View Purchased Documents

Profile Information

» Communications Preferences

» Profession and Education

» Technical Interests

Need Help?

» **US & Canada:** +1 800 678 4333

» **Worldwide:** +1 732 981 0060

» Contact & Support

About IEEE *Xplore* | Contact Us | Help | Accessibility | Terms of Use | Nondiscrimination Policy | Sitemap | Privacy & Opting Out of Cookies

A not-for-profit organization, IEEE is the world's largest technical professional organization dedicated to advancing technology for the benefit of humanity.

© Copyright 2025 IEEE - All rights reserved. Use of this web site signifies your agreement to the terms and conditions.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0171

IEEE.org IEEE *Xplore* IEEE SA IEEE Spectrum More Sites Subscribe Donate Cart Welcome Charles Dougherty Sign Out

Browse My Settings Help Institutional Sign In

Institutional Sign In

All

Search within Publication

ADVANCED SEARCH

Quick Links
Search for Upcoming Conferences
IEEE Publication Recommender
IEEE Author Center

International Conference on Dependable Systems and Networks Workshops (DSN-W)

Proceedings

The proceedings of this conference will be available for purchase through Curran Associates.

Dependable Systems and Networks Workshops (DSN-W), 2012 IEEE/IFIP 42nd (PRT)

Print on Demand **Purchase at Partner**

Copy Persistent Link Browse Title List Sign up for Conference Alerts

Proceedings All Proceedings Popular

IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012) doi
25-28 June 2012

DOI: 10.1109/DSN-W21105.2012

Search within results

Items Per Page Export Email Selected Results

Showing **1-25** of **49**

Filter

Sort Sequence Sort Email

Refine

Author

Affiliation

Quick Links
Search for Upcoming Conferences
IEEE Publication Recommender
IEEE Author Center

Jean-Claude Laprie award in dependable computing
Publication Year: 2012 , Page(s): 1 - 1

Jean-Claude Laprie award in dependable computing
IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)
Year: 2012

Message from the DCCS program chair
Philip Koopman
Publication Year: 2012 , Page(s): 1 - 1

Proceedings

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0172

The proceedings of this conference will be available for purchase through Curran Associates.

Dependable Systems and Networks Workshops (DSN-W), 2012 IEEE/IFIP 42nd (PRT)

Print on Demand **Purchase at Partner**

Message from the DCCS program chair

Philip Koopman
IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)
Year: 2012

Organizers DSN 2012
Publication Year: 2012 , Page(s): 1 - 5

© Organizers DSN 2012

IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)
Year: 2012

Message from the PDS program chair
Michel Cukier
Publication Year: 2012 , Page(s): 1 - 1

© Message from the PDS program chair

Michel Cukier
IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)
Year: 2012

Sponsors
Publication Year: 2012 , Page(s): 1 - 1

© Sponsors

IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)
Year: 2012

William C. Carter Award
Collin Mulliner
Publication Year: 2012 , Page(s): 1 - 1

© William C. Carter Award

Collin Mulliner
IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)
Year: 2012

Message from the general chair
Robert Swarz
Publication Year: 2012 , Page(s): 1 - 1

© Message from the general chair

Robert Swarz
IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)
Year: 2012

WORCS committees

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0173

Publication Year: 2012 , Page(s): 1 - 2

WORCS committees

IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)
Year: 2012

Workshop on open resilient human-aware Cyber-physical systems
Mohamed Kaâniche; Michael Harrison; Hermann Kopetz; Daniel Siewiorek
Publication Year: 2012 , Page(s): 1 - 2

Abstract **HTML**

Workshop on open resilient human-aware Cyber-physical systems

Mohamed Kaâniche; Michael Harrison; Hermann Kopetz; Daniel Siewiorek
IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)
Year: 2012

Invited talk: Challenges in Medical Cyber-Physical Systems
Insup Lee
Publication Year: 2012 , Page(s): 1 - 1
Cited by: Papers (1)

Abstract **HTML**

Invited talk: Challenges in Medical Cyber-Physical Systems

Insup Lee
IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)
Year: 2012

Invited talk: Virtual coaches in health care
Daniel P. Siewiorek
Publication Year: 2012 , Page(s): 1 - 1
Cited by: Papers (1)

Abstract **HTML**

Invited talk: Virtual coaches in health care

Daniel P. Siewiorek
IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)
Year: 2012

Invited talk: A model and simulation for user-centric automation devices and systems
Jane W. S. Liu
Publication Year: 2012 , Page(s): 1 - 1

Abstract **HTML**

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0174

Invited talk: A model and simulation for user-centric automation devices and systems

Jane W. S. Liu
IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)
Year: 2012

Smart checklists for human-intensive medical systems
George S. Avrunin; Lori A. Clarke; Leon J. Osterweil; Julian M. Goldman; Tracy Rausch
Publication Year: 2012 , Page(s): 1 - 6
Cited by: Papers (15)

Abstract **HTML**

Smart checklists for human-intensive medical systems

George S. Avrunin; Lori A. Clarke; Leon J. Osterweil; Julian M. Goldman; Tracy Rausch
IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)
Year: 2012

Towards resiliency in embedded medical monitoring devices
Homa Alemzadeh; Catello Di Martino; Zhanpeng Jin; Zbigniew T. Kalbarczyk; Ravishankar K. Iyer
Publication Year: 2012 , Page(s): 1 - 6
Cited by: Papers (6)

Abstract **HTML**

Towards resiliency in embedded medical monitoring devices

Homa Alemzadeh; Catello Di Martino; Zhanpeng Jin; Zbigniew T. Kalbarczyk; Ravishankar K. Iyer
IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)
Year: 2012

A rigorous approach to the design of resilient cyber-physical systems through co-simulation
John Fitzgerald; Ken Pierce; Carl Gamble
Publication Year: 2012 , Page(s): 1 - 6
Cited by: Papers (5)

Abstract **HTML**

A rigorous approach to the design of resilient cyber-physical systems through co-simulation

John Fitzgerald; Ken Pierce; Carl Gamble
IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)
Year: 2012

Physical attack protection with human-secure virtualization in data centers
Jakub Szefer; Pramod Jamkhedkar; Yu-Yuan Chen; Ruby B. Lee
Publication Year: 2012 , Page(s): 1 - 6
Cited by: Papers (5)

Abstract **HTML**

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0175

Physical attack protection with human-secure virtualization in data centers

Jakub Szefer; Pramod Jamkhedkar; Yu-Yuan Chen; Ruby B. Lee
IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)
Year: 2012

MILS-related information flow control in the avionic domain: A view on security-enhancing software architectures
Kevin Müller; Michael Paulitsch; Sergey Tverdyshev; Holger Blasum
Publication Year: 2012 , Page(s): 1 - 6
Cited by: Papers (6) | Patents (2)

Abstract **HTML**

MILS-related information flow control in the avionic domain: A view on security-enhancing software architectures

Kevin Müller; Michael Paulitsch; Sergey Tverdyshev; Holger Blasum
IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)
Year: 2012

Panel: Challenges and research directions in resilient cyber-physical systems
Publication Year: 2012 , Page(s): 1 - 1

Abstract **HTML**

Panel: Challenges and research directions in resilient cyber-physical systems

IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)
Year: 2012

Chaotic-identity maps for robustness estimation of exascale computations
Nageswara S. V. Rao
Publication Year: 2012 , Page(s): 1 - 6
Cited by: Papers (2)

Abstract **HTML**

Chaotic-identity maps for robustness estimation of exascale computations

Nageswara S. V. Rao
IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)
Year: 2012

Asynchronous checkpoint migration with MRNet in the Scalable Checkpoint / Restart Library
Kathryn Mohror; Adam Moody; Bronis R. de Supinski
Publication Year: 2012 , Page(s): 1 - 6
Cited by: Papers (2)

Abstract **HTML**

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0176

Asynchronous checkpoint migration with MRNet in the Scalable Checkpoint / Restart Library

Kathryn Mohror; Adam Moody; Bronis R. de Supinski
IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)
Year: 2012

Does partial replication pay off?
Jon Stearley; Kurt Ferreira; David Robinson; Jim Laros; Kevin Pedretti; Dorian Arnold; Patrick Bridges; Rolf Riesen
Publication Year: 2012 , Page(s): 1 - 6
Cited by: Papers (17)

Abstract **HTML**

Does partial replication pay off?

Jon Stearley; Kurt Ferreira; David Robinson; Jim Laros; Kevin Pedretti; Dorian Arnold; Patrick Bridges; Rolf Riesen
IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)
Year: 2012

Energy considerations in checkpointing and fault tolerance protocols
M. el Mehdi Diouri; Olivier Glück; Laurent Lefevre; Franck Cappello
Publication Year: 2012 , Page(s): 1 - 6
Cited by: Papers (21)

Abstract **HTML**

Energy considerations in checkpointing and fault tolerance protocols

M. el Mehdi Diouri; Olivier Glück; Laurent Lefevre; Franck Cappello
IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)
Year: 2012

A programming model for resilience in extreme scale computing
Saurabh Hukerikar; Pedro C. Diniz; Robert F. Lucas
Publication Year: 2012 , Page(s): 1 - 6
Cited by: Papers (10)

Abstract **HTML**

A programming model for resilience in extreme scale computing

Saurabh Hukerikar; Pedro C. Diniz; Robert F. Lucas
IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)
Year: 2012

ROSE::FTTransform - A source-to-source translation framework for exascale fault-tolerance research
Jacob Lidman; Daniel J. Quinlan; Chunhua Liao; Sally A. McKee
Publication Year: 2012 , Page(s): 1 - 6
Cited by: Papers (22) | Patents (1)

Abstract **HTML**

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0177

ROSE::FTTransform - A source-to-source translation framework for exascale fault-tolerance research

Jacob Lidman; Daniel J. Quinlan; Chunhua Liao; Sally A. McKee
IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)
Year: 2012

A message-logging protocol for multicore systems
Esteban Meneses; Xiang Ni; Laxmikant V. Kalé
Publication Year: 2012 , Page(s): 1 - 6
Cited by: Papers (10)

Abstract **HTML**

A message-logging protocol for multicore systems

Esteban Meneses; Xiang Ni; Laxmikant V. Kalé
IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)
Year: 2012

Load More

1 2 >

IEEE Personal Account
CHANGE USERNAME/PASSWORD

Purchase Details
PAYMENT OPTIONS
VIEW PURCHASED DOCUMENTS

Profile Information
COMMUNICATIONS PREFERENCES
PROFESSION AND EDUCATION
TECHNICAL INTERESTS

Need Help?
US & CANADA: +1 800 678 4333
WORLDWIDE: +1 732 981 0060
CONTACT & SUPPORT

Follow

About IEEE *Xplore* | Contact Us | Help | Accessibility | Terms of Use | Nondiscrimination Policy | IEEE Ethics Reporting | Sitemap | IEEE Privacy Policy
A public charity, IEEE is the world's largest technical professional organization dedicated to advancing technology for the benefit of humanity.

© Copyright 2025 IEEE - All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies.

IEEE Account
» Change Username/Password
» Update Address
Purchase Details
» Payment Options
» Order History
» View Purchased Documents
Profile Information
» Communications Preferences
» Profession and Education

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0178

» Technical Interests

Need Help?

» **US & Canada:** +1 800 678 4333

» **Worldwide:** +1 732 981 0060

» Contact & Support

About IEEE *Xplore* | Contact Us | Help | Accessibility | Terms of Use | Nondiscrimination Policy | Sitemap | Privacy & Opting Out of Cookies

A not-for-profit organization, IEEE is the world's largest technical professional organization dedicated to advancing technology for the benefit of humanity.
© Copyright 2025 IEEE - All rights reserved. Use of this web site signifies your agreement to the terms and conditions.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0179

IEEE.org IEEE *Xplore* IEEE SA IEEE Spectrum More Sites Subscribe Donate Cart Welcome Charles Dougherty Sign Out

Browse My Settings Help Institutional Sign In

Institutional Sign In

All

Search within Publication

ADVANCED SEARCH

Quick Links
Search for Upcoming Conferences
IEEE Publication Recommender
IEEE Author Center

Proceedings

Browse Conferences > International Conference on De... > IEEE/IFIP International Confer...

International Conference on Dependable Systems and Networks Workshops (DSN-W)

The proceedings of this conference will be available for purchase through Curran Associates.

Dependable Systems and Networks Workshops (DSN-W), 2012 IEEE/IFIP 42nd (PRT)

Print on Demand **Purchase at Partner**

Copy Persistent Link Browse Title List Sign up for Conference Alerts

Proceedings All Proceedings Popular

IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012) doi
25-28 June 2012

DOI: 10.1109/DSN-W21105.2012

Search within results

Items Per Page Export Email Selected Results

Showing **26-49** of **49**

Filter Sort Sequence Sort Email

Refine

Author

Affiliation

Quick Links
Search for Upcoming Conferences
IEEE Publication Recommender
IEEE Author Center

Proceedings

An evaluation of difference and threshold techniques for efficient checkpoints
Sean Hogan; Jeff R. Hammond; Andrew A. Chien
Publication Year: 2012 , Page(s): 1 - 6
Cited by: Papers (4)

Abstract **HTML**

An evaluation of difference and threshold techniques for efficient checkpoints
Sean Hogan; Jeff R. Hammond; Andrew A. Chien
IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)
Year: 2012

On the complexity of scheduling checkpoints for computational workflows
Yves Robert; Frédéric Vivien; Dounia Zaidouni
Publication Year: 2012 , Page(s): 1 - 6

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0180

The proceedings of this conference will be available for purchase through Curran Associates.

Dependable Systems and Networks Workshops (DSN-W), 2012 IEEE/IFIP 42nd (PRT)

Print on Demand **Purchase at Partner**

Cited by: Papers (7)

Abstract **HTML**

On the complexity of scheduling checkpoints for computational workflows

Yves Robert; Frédéric Vivien; Dounia Zaidouni
IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)
Year: 2012

Design and implementation of a hardware checkpoint/restart core
Ashwin A. Mendon; Ron Sass; Zachary K. Baker; Justin L. Tripp
Publication Year: 2012 , Page(s): 1 - 6
Cited by: Papers (3)

Abstract **HTML**

Design and implementation of a hardware checkpoint/restart core

Ashwin A. Mendon; Ron Sass; Zachary K. Baker; Justin L. Tripp
IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)
Year: 2012

A scalable double in-memory checkpoint and restart scheme towards exascale
Gengbin Zheng; Xiang Ni; Laxmikant V. Kalé
Publication Year: 2012 , Page(s): 1 - 6
Cited by: Papers (72)

Abstract **HTML**

A scalable double in-memory checkpoint and restart scheme towards exascale

Gengbin Zheng; Xiang Ni; Laxmikant V. Kalé
IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)
Year: 2012

6th workshop on recent advances in intrusion tolerance and reSilience (WRAITS 2012)
Ilir Gashi; O. Patrick Kreidl
Publication Year: 2012 , Page(s): 1 - 2
Cited by: Papers (1)

Abstract **HTML**

6th workshop on recent advances in intrusion tolerance and reSilience (WRAITS 2012)

Ilir Gashi; O. Patrick Kreidl
IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)
Year: 2012

Avoiding common security flaws in composed service-oriented systems
Michael Atighetchi; Partha Pal; Joseph Loyall; Asher Sinclair
Publication Year: 2012 , Page(s): 1 - 6

Abstract **HTML**

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0181

Avoiding common security flaws in composed service-oriented systems

Michael Atighetchi; Partha Pal; Joseph Loyall; Asher Sinclair
IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)
Year: 2012

Improving resilience of SOA services along space-time dimensions
Quyen L. Nguyen; Arun Sood
Publication Year: 2012 , Page(s): 1 - 6
Cited by: Papers (2)

Abstract **HTML** ©

Improving resilience of SOA services along space-time dimensions

Quyen L. Nguyen; Arun Sood
IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)
Year: 2012

People-oriented cyber security
Greg Frazier
Publication Year: 2012 , Page(s): 1 - 1

Abstract **HTML** ©

People-oriented cyber security

Greg Frazier
IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)
Year: 2012

A model for security analysis of smart meters
Farid Molazem Tabrizi; Karthik Pattabiraman
Publication Year: 2012 , Page(s): 1 - 6
Cited by: Papers (18)

Abstract **HTML** ©

A model for security analysis of smart meters

Farid Molazem Tabrizi; Karthik Pattabiraman
IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)
Year: 2012

On limitations of using cloud storage for data replication
Christian Cachin; Birgit Junker; Alessandro Sorniotti
Publication Year: 2012 , Page(s): 1 - 6
Cited by: Papers (4)

Abstract **HTML** ©

On limitations of using cloud storage for data replication

Christian Cachin; Birgit Junker; Alessandro Sorniotti
IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)
Year: 2012

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0182

Towards incorporating human intelligence into online security solutions
Saman Zonouz; Robin Berthier; Negin Arhami
Publication Year: 2012 , Page(s): 1 - 2

Abstract **HTML**

Towards incorporating human intelligence into online security solutions

Saman Zonouz; Robin Berthier; Negin Arhami
IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)
Year: 2012

A security analysis of Amazon's Elastic Compute Cloud service
Engin Kirda
Publication Year: 2012 , Page(s): 1 - 1
Cited by: Patents (1)

Abstract **HTML**

A security analysis of Amazon's Elastic Compute Cloud service

Engin Kirda
IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)
Year: 2012

The TClouds architecture: Open and resilient cloud-of-clouds computing
Paulo Verissimo; Alysson Bessani; Marcelo Pasin
Publication Year: 2012 , Page(s): 1 - 6
Cited by: Papers (28)

Abstract **HTML**

The TClouds architecture: Open and resilient cloud-of-clouds computing

Paulo Verissimo; Alysson Bessani; Marcelo Pasin
IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)
Year: 2012

Toward a high availability cloud: Techniques and challenges
Cuong Pham; Phuong Cao; Zbigniew Kalbarczyk; Ravishankar K. Iyer
Publication Year: 2012 , Page(s): 1 - 6
Cited by: Papers (14)

Abstract **HTML**

Toward a high availability cloud: Techniques and challenges

Cuong Pham; Phuong Cao; Zbigniew Kalbarczyk; Ravishankar K. Iyer
IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)
Year: 2012

Dependability as a cloud service - a modular approach
Jan S. Rellermeyer; Saurabh Bagchi
Publication Year: 2012 , Page(s): 1 - 6
Cited by: Papers (4)

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0183

Abstract **HTML**

Dependability as a cloud service - a modular approach

Jan S. Rellermeyer; Saurabh Bagchi
IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)
Year: 2012

Osprey: Operating system for predictable clouds
Jan Sacha; Jeff Napper; Sape Mullender; Jim McKie
Publication Year: 2012 , Page(s): 1 - 6
Cited by: Papers (6)

Abstract **HTML**

Osprey: Operating system for predictable clouds

Jan Sacha; Jeff Napper; Sape Mullender; Jim McKie
IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)
Year: 2012

Defending against VM rollback attack
Yubin Xia; Yutao Liu; Haibo Chen; Binyu Zang
Publication Year: 2012 , Page(s): 1 - 5
Cited by: Papers (13) | Patents (1)

Abstract **HTML**

Defending against VM rollback attack

Yubin Xia; Yutao Liu; Haibo Chen; Binyu Zang
IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)
Year: 2012

TinyChecker: Transparent protection of VMs against hypervisor failures with nested virtualization
Cheng Tan; Yubin Xia; Haibo Chen; Binyu Zang
Publication Year: 2012 , Page(s): 1 - 6
Cited by: Papers (12) | Patents (1)

Abstract **HTML**

TinyChecker: Transparent protection of VMs against hypervisor failures with nested virtualization

Cheng Tan; Yubin Xia; Haibo Chen; Binyu Zang
IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)
Year: 2012

Utilizing linear subspaces to improve cloud security
David Zage; James Obert
Publication Year: 2012 , Page(s): 1 - 6
Cited by: Papers (3)

Abstract **HTML**

Utilizing linear subspaces to improve cloud security

David Zage; James Obert
IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)
Year: 2012

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0184

Evidence of log integrity in policy-based security monitoring
Mirko Montanari; Jun Ho Huh; Derek Dagit; Rakesh B. Bobba; Roy H. Campbell
Publication Year: 2012 , Page(s): 1 - 6
Cited by: Papers (7)

Abstract **HTML**

Evidence of log integrity in policy-based security monitoring

Mirko Montanari; Jun Ho Huh; Derek Dagit; Rakesh B. Bobba; Roy H. Campbell
IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)
Year: 2012

TRONE: Trustworthy and Resilient Operations in a Network Environment
António Casimiro; Paulo Veríssimo; Diego Kreutz; Filipe Araujo; Raul Barbosa; Samuel Neves; Bruno Sousa; Marília Curado; Carlos Silva; Rajeev Gandhi; Priya Narasimhan
Publication Year: 2012 , Page(s): 1 - 6
Cited by: Papers (3)

Abstract **HTML**

TRONE: Trustworthy and Resilient Operations in a Network Environment

António Casimiro; Paulo Veríssimo; Diego Kreutz; Filipe Araujo; Raul Barbosa; Samuel Neves; Bruno Sousa; Marília Curado; Carlos Silva; Rajeev Gandhi; Priya Narasimhan
IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)
Year: 2012

An independent verification of errors and vulnerabilities in SaaS cloud
Rajeshwari Ganesan; Santonu Sarkar; Naveen Tewari
Publication Year: 2012 , Page(s): 1 - 6
Cited by: Papers (2)

Abstract **HTML**

An independent verification of errors and vulnerabilities in SaaS cloud

Rajeshwari Ganesan; Santonu Sarkar; Naveen Tewari
IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)
Year: 2012

A study of fault-tolerance characteristics of data center networks
Yang Liu; Dong Lin; Jogesh Muppala; Mounir Hamdi
Publication Year: 2012 , Page(s): 1 - 6
Cited by: Papers (9)

Abstract **HTML**

A study of fault-tolerance characteristics of data center networks

Yang Liu; Dong Lin; Jogesh Muppala; Mounir Hamdi
IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)
Year: 2012

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0185

Models for dependability and sustainability analysis of data center cooling architectures
Gustavo Callou; Paulo Maciel; Dietmar Tutsch; Julian Araújo
Publication Year: 2012 , Page(s): 1 - 6
Cited by: Papers (22)

Abstract HTML

Models for dependability and sustainability analysis of data center cooling architectures

Gustavo Callou; Paulo Maciel; Dietmar Tutsch; Julian Araújo
IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)
Year: 2012

Load More

< 1 2

IEEE Personal Account
CHANGE USERNAME/PASSWORD

Purchase Details
PAYMENT OPTIONS
VIEW PURCHASED DOCUMENTS

Profile Information
COMMUNICATIONS PREFERENCES
PROFESSION AND EDUCATION
TECHNICAL INTERESTS

Need Help?
US & CANADA: +1 800 678 4333
WORLDWIDE: +1 732 981 0060
CONTACT & SUPPORT

Follow

About IEEE *Xplore* | Contact Us | Help | Accessibility | Terms of Use | Nondiscrimination Policy | IEEE Ethics Reporting | Sitemap | IEEE Privacy Policy
A public charity, IEEE is the world's largest technical professional organization dedicated to advancing technology for the benefit of humanity.

© Copyright 2025 IEEE - All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies.

IEEE Account
» Change Username/Password
» Update Address
Purchase Details
» Payment Options
» Order History
» View Purchased Documents
Profile Information
» Communications Preferences
» Profession and Education
» Technical Interests
Need Help?
» **US & Canada:** +1 800 678 4333

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0186

» **Worldwide:** +1 732 981 0060

» Contact & Support

About IEEE *Xplore* | Contact Us | Help | Accessibility | Terms of Use | Nondiscrimination Policy | Sitemap | Privacy & Opting Out of Cookies

A not-for-profit organization, IEEE is the world's largest technical professional organization dedicated to advancing technology for the benefit of humanity.
© Copyright 2025 IEEE - All rights reserved. Use of this web site signifies your agreement to the terms and conditions.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0187

Dependability as a Cloud Service - A Modular Approach

Jan S. Rellermeyer[1]
[1]*IBM Research,*
Austin Research Lab,
Austin, TX 78758
rellermeyer@us.ibm.com

Saurabh Bagchi[1,2]
[2]*Purdue University,*
School of Electrical and Computer Engineering,
West Lafayette, IN 47907
sbagchi@purdue.edu, sbagchi@us.ibm.com

***Abstract*—Failures of services on cloud platforms are only to be expected. To deal with such failures, one is naturally inclined to use the traditional measure of replication. However, replication of services on distributed cloud platforms poses several challenges that are not well met by today's Java middleware systems. These challenges are the need to isolate state in the application components so that easy migration and recovery are possible and the requirement for client transparency when dealing with different replicated service instances. For example, Java Enterprise Edition (JEE) makes it difficult to have transparent replication of services due to the above two reasons plus the fine-grained nature of interactions between its components (the Enterprise Java Beans). In this paper, we show parts of the design of OSGi, a specification defining a dynamic component system in Java, that make it suitable for the above task. We then propose two extensions to OSGi which will allow exposing and exporting application component state and transparent invocation of service instances. These two together can enable easy replication and recovery from failures in cloud environments. We show through experiments that our prototype can migrate a failed service quickly enough to a new machine so that a client experiences only a moderate increase in service invocation time during system recovery.**

I. Introduction

A decision to move away from privately owned computer resources and towards cloud computing is affected by a variety of factors, dependability being one of them. Dependability is usually defined through its attributes of availability, reliability, safety, integrity, and maintainability [1]. In a complex system, the overall dependability of the system depends on the corresponding attributes of its components. What changes in cloud computing is that one important component—the platform—is no longer under one's own control. Depending on the type of the cloud service, the platform may mean one of several things; at least it includes the physical machine but may also include higher layers like a complete runtime system. This arguably outsources part of the maintenance problem while introducing new challenges. Most importantly, the underlying physical resources (i.e., the machines and the network) are shared between multiple customers and applications and cannot be actively managed. This is often perceived to be a threat to the overall dependability [2], [3], [4].

Typical ways of mitigating these threats in a cloud computing environment revolve around introducing redundancy into the system, especially for increased availability and reliability. Since adverse incidents can always render a single instance of a functional unit unusable, the hope is that by creating multiple copies of the component a sufficient number of copies always survive to deliver the service without degradation. This idea is neither new nor specific to cloud computing and has been successfully applied many many times in earlier distributed systems such as clusters or grids [5], [6]. In cloud computing, however, the effectiveness of redundancy benefits from two particular factors. First, the elasticity of the resource fabric makes redundancy more agile and more affordable. This means that it is possible to acquire additional resources for increasing the degree of redundancy in response to perceived threats much more easily than it can be done in traditional hosted environments. Second, the resource fabric is typically globally distributed, thereby allowing intelligent placement to guard against high-impact outages that would be much harder to circumvent in a hosted environment. By spanning multiple geographic zones, a system can survive even an outage of an entire data center that could be caused by a natural disaster or a far-reaching power outage.

Unfortunately, not every component of a complex system can easily profit from redundancy. Components that have state in them cannot be easily split up for purposes of replicating the parts to different replication degrees and executed on different machines. There have been prior attempts to address this by structuring web services such that most of the components are stateless and the few stateful components expose their state in an orderly manner. Thereby, it can be handled through special, and more expensive, mechanisms such as database replication. However, we discern that this kind of structuring of applications is becoming harder to do. Web services are becoming contextualized by various factors including the user, the client environment, the results of prior requests, and interactions with coordinating and collaborating web services. For example, with the advent of social networks, the services are contextualized by the user profiles and for a rich interactive user experience, the

978-1-4673-2266-9/12/$31.00 ©2012 IEEE

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0188

state (in this case, the profile) is driving the majority of the functionality. *Therefore, our position is that existing mechanisms for structuring applications for purposes of reliability will no longer work with emerging web services, that are being hosted on the cloud.*

The effectiveness of elasticity depends to a high degree on the coupling of components. Systems with fine granular components tend to suffer from a high degree of entanglement, hence limiting the flexibility of selectively replicating components since all tightly-coupled components usually have to reside on the same host. Consider for example, Enterprise Java Beans running on a JEE middleware are typically structured to have final granularity and with frequent invocations between them, both of which highlight the challenges of replication for cloud environments.

In this paper, we discuss how a dependability service can be built on the OSGi framework. OSGi is a set of specifications [7] that defines a dynamic component system for Java. These specifications enable a development model where applications are *dynamically* composed of many different reusable components. The OSGi specifications enable components to hide their implementations from other components while communicating through services, which are objects that are specifically shared between components. We show why a module system for Java, like OSGi, makes our goal simpler to achieve and then show how the dependability service can be architected using features already provided by OSGi. We propose an extension to the OSGi standard that acts as a dependability service, which we call CLOUDDEP. The service provides the following dependability properties:

1) Re-deployment: Migrating the application to a new physical machine or a different data center when the underlying platform fails.
2) Clustering: Eliminating single points of failure in the system by replicating functional units.
3) Elasticity and Load-balancing: Distributing the load to a flexible pool of resources to avoid overloading single instances.

II. SOFTWARE MODULARITY AND DEPENDABILITY

Software modules are self-contained functional units of encapsulation. They contain data and functions, objects, or other entities of the programming languages used to implement them. Ideally, the correctness of a modular system depends only on the correctness of its modules and the correctness of a single module does not depend on other modules [8]. The requirement of being self-contained, however does not automatically imply full isolation. Many implementations of module technology allow modules to be *declaratively self-contained*, i.e., they not only rely on their own content but can additionally declare dependencies to other modules and have access to their accessible content. In such setups, the effects of errors occurring in a single module are still restricted to cause faults in this module or any module that has a declared dependency but not in unrelated modules.

This property has been successfully exploited to increase the fault-tolerance of systems. For instance, in the Actor model [9] the modular units are fully isolated and can only communicate through explicit message passing.

Micro Kernel Operating Systems [10] restrict themselves to implement only the minimally necessary functionality in the kernel and outsource the remaining responsibilities into separate user-space modules. Thereby, faults in one module cannot affect other modules or the privileged kernel.

Microreboots [11] describe a methodology for recovering a system by rebooting only a subset of affected components, in the concrete example Enterprise Java Beans (EJBs). The design approach is to keep application state carefully parceled away in separate state stores which are only accessed through well-defined interfaces. Thus, the application components can be recovered by rebooting them (since the rebooting is fast in the absence of state, it is called microrebooting), while the state store can persist across reboots. This approach shares some similarities with our proposed approach. However, in order to be micro-rebootable, the components need to be well-isolated and stateless except for externalized application state kept in specialized state stores. Such compartmentalization is non-trivial to do in the context of JEE because of the reasons mentioned earlier (tight coupling and fine granular decomposition).

III. OSGI

A. OSGi Background

OSGi is a set of standards for modularity on the Java virtual machine released and maintained by the OSGi Alliance. Originally, it was developed by several major vendors to solve an availability problem on embedded home gateway machines [12]. The intention was that multiple software components should be able to co-exist on the same hardware while each of the software packages needs to run continuously and over long periods of time, even in scenarios that require periodic maintenance updates to the software. To enable dynamic updates without stoppage, it aimed at overcoming the traditional entanglement of small-granular objects and formed larger modules that can be managed individually and composed to form applications. OSGi defines a unit of modularity as the *Bundle*, which is a deployable Java JAR file enriched with additional meta-data. Most importantly, this meta-data describes which packages are exported, i.e., accessible to other bundles, and which packages it needs to import from other bundles. Recently, OSGi has successfully entered the domain of enterprise software and most major Java application servers (e.g., IBM WebSphere, JBoss, Oracle WebLogic, Spring Dynamic Module Platform) are by now internally based on the OSGi technology.

The OSGi runtime environment, the *Framework*, provides the user with mechanisms for installing and controlling the life-cycle of bundles at runtime. Applications are typically composed out of many different bundles. These bundles can interact in a tightly-coupled way by depending on packages provided by other bundles. *Tight coupling* in this context means that the dependencies are required to resolve the bundle and failure to provide the dependencies renders the bundle inoperable. Another way of interaction is through loosely-coupled services. *Loose coupling* means that service consumers only rely on the service interfaces under which the service has been registered with the OSGi framework but not on any knowledge of implementation details. Bundles can consequently query the framework for available services and retrieve the service instance, which is an ordinary Java object in OSGi. The result of a query is either a reference, a set of references from which the client can pick the best matching service, or the information that no such service is available. Since the consumers only rely on the service interfaces but not on any concrete implementation, bundles can potentially operate even in the absence of certain services. OSGi applications are expected to actively deal with dynamism in the setup. For instance, a service can certainly fulfill its purpose in the absence of a log service but if it has logging support, e.g., for debugging, it should enable it whenever a log service becomes available and disable it when the service becomes unavailable.

The modularity in OSGi solves a major issue with creating elastic applications for the cloud. Cloud resources are inherently dynamic in their nature and undergo frequent changes either due to explicit management operations (adding and removing resources) or due to their volatility (sharing effects, failures). Therefore, we believe that monolithic software incapable of dynamic adaptation cannot effectively run in such an environment. In fact, the compositional approach of modular software—traditionally applied to a single runtime system—is key to building scalable and dependable systems across a varying set of machines in the cloud [13], [14]. OSGi particularly facilitates composition and localized behavior by separating tightly coupled and loosely-coupled interaction between modules into two different concepts: package dependencies and services, respectively. Thereby, when services are used as a design element, the resulting systems are less entangled and more flexible.

OSGi was originally designed for managing software on a single Java virtual machine. In prior work, we have shown that the OSGi model can be used as an application model for building distributed systems by turning the services into potential boundaries of distribution [15] and adding simple support for creating redundancy by clustering stateless services on multiple machines [16]. However, little attention has so far been paid to the problem of managing stateful services in distributed deployments like clusters or clouds.

B. How OSGi Configuration Admin Enables Restarts

A traditional OSGi framework (running on a single JVM) already supports restarts of the runtime systems as well as a form of partial restart for bundle updates. This means that whenever a bundle has been updated and the operator of the framework requests a refresh of the system, only those bundles immediately affected by the update (this is the bundle itself and all bundles relying on the updated bundle through tightly-coupled package dependencies) are restarted whereas the remainder of the system is unaffected. However, only the compositional state of the application is preserved across restarts. This is the information about which bundles have been installed when the framework was last running and in which life-cycle state each individual bundle has been in (installed, resolved, starting, active, stopping, uninstalled). Every bundle then gets started again but is itself responsible for restoring any internal state, which includes services that it has bound to.

The only platform support in traditional OSGi for externalizing bundle state is the Configuration Admin service, an optional service described in the OSGi Compendium Specifications [17]. The configuration admin is a service in the system capable of managing and persisting configuration data in a central location. Services that want either to be externally configurable or want to persist their configuration data across restarts can register as a *Managed Service*. This expresses its intent to receive configuration updates and gives the system an interface through which it can push updates to the service. Every managed service is expected to identify itself though a persistent identifier (PID) and announce this PID with its initial registration. The config admin uses the PID of a service as a primary key for associating configuration data to services. Configuration events and changes are signaled to the managed service asynchronously, regardless of whether they are used by the service itself (as illustrated in Figure 1) or an external bundle.

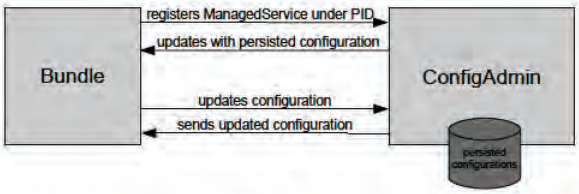


Figure 1. Interaction between Bundle and Configuration Admin

In the context of cloud computing, the most interesting and relevant property of the config admin is the concept of a service identity. When migrating a service to a new machine, it has to be assured that the newly created service instance can fully take over the role of the previous instance. Therefore, stateful services implicitly build up their own identity based on their internal state and their

history. Migration and failover can be considered a "micro-reboot" of the service on a different machine. If all state could be stored as configuration data then a distributed version of the configuration admin would turn managed services into migratable services just as the ConfigAdmin allows stateful services to become restartable. In practice, the storage capacity of the configuration admin and its unclear consistency model are prohibitive to this approach. For instance, a configuration update always fully overwrites any previous configuration so that when two writers are updating disjoint parts of the configuration concurrently the last writer wins and the update from the earlier writer is lost.

However, just as systems like Zookeeper [18] are used in cloud deployments to manage configuration, a cloud-scale version of an OSGi ConfigAdmin can be used to dynamically bind service instances to a richer instance of storage to externalize the service state, e.g., a database system or a key-value store. The challenge of building such a distributed ConfigAdmin for cloud deployments is to coordinate it with the distribution of services. It is not feasible to have a single scope for configurations across all nodes since this would prohibit multiple redundant copies of the same service. When a service needs to be migrated to a different machine, however, the configuration has to be migrated with the service.

C. How OSGi Remote Service Admin Enables Non-local Execution

The Remote Service Admin aims at integrating distribution into the OSGi platform and has been part of the OSGi Enterprise Specifications [19]. This service encapsulates the mechanism for importing remote services into the scope of a local OSGi framework and exporting local OSGi services through distribution providers. When a remote service is imported, a local proxy for the remote service is created and associated with the distribution provider. Some distribution providers like our R-OSGi [15] system are able to operate transparently, i.e., neither client nor service have to be rewritten to operate in a remote setup.

The API of the remote service admin is entirely imperative. Given a known URI or other identifier, the admin provisions the local proxy and handles binding to the remote service reachable through this address, as long as the service is reachable. Finding a service in the network and re-binding in the event of a failure is handled by different components. All policy decisions are encapsulated in a Topology Manager which, e.g., is able to detect clients that are querying the local framework for services and make use of service discovery protocols to find matching services. However, the specifications do not mandate any particular functionality of the topology manager but only specify the interfaces through which a possible topology manager can interact.

For a cloud setup with dependability requirements, the presence of an appropriate topology manager is paramount.

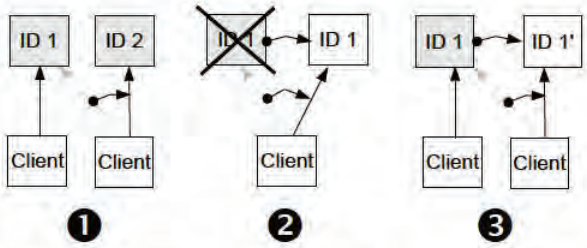


Figure 2. Different Use Cases for a Dependability Service. 1: Load balancing, 2: Failover, 3: Elastic Scalability.

In the following section, we describe our service CLOUDDEP that implements policies to ensure availability of services by acting as both a topology manager and a distributed configuration admin.

IV. TOWARDS A DEPENDABILITY SERVICE

The new service (that we call CLOUDDEP) is designed to turn OSGi services into elastic units of deployment which can be migrated and replicated across multiple cloud nodes to increase the availability and reliability of the administered service. All that is required from the administered service is that it persists all of its internal state either directly or indirectly through configurations and listens to configuration updates by implementing the Managed Service interface of the configuration admin specification. CLOUDDEP acts as a global entity in the system (which can be implemented by a centralized backend or a distributed architecture) and takes over the role of a local configuration admin and a local topology manager on each node of the cloud deployment. It also monitors the service invocations to detect failures. We illustrate the features of CLOUDDEP through three usage cases.

A. Load Balancing

We consider a simple setup with a set of identical (possibly stateful) remote service and a set of service clients as illustrated by case ❶ in Figure 2. Thanks to its facet of being a topology manager, CLOUDDEP has knowledge of the remote service relationship and was involved in the setup of the binding. It had detected the local service requests of the client bundles (e.g., through service lookup hooks as described in the remote service admin specification) and has instructed the distribution provider to create the binding to one of the known remote service instances. Furthermore, it has, either due to cooperation of the distribution provider or by wrapping its created service proxy, instrumented the local proxy to gather information about the frequencies of service invocation and response times. In the event that one remote service instance becomes overloaded, detectable by an increase of invocations and an increase of the response time, CLOUDDEP can re-bind a client to a different service

instance, thereby re-distributing the load among the service instances.

B. Failover

We now consider a setup with one or more clients and a single remote service. In the event of a service failure (e.g., an uncaught runtime exception or the node hosting the remote service becoming unavailable) CLOUDDEP has captured the identity of the faulty service through its configuration admin facet. It can thereby instantiate a new service by starting the corresponding bundle on a new node and inject the identity from the failed service into the new instance, as shown in case ❷. This process is akin to what a microreboot would do on a single node but here applied to a distributed setup. By applying the same rebinding as done for load balancing, the existing client will be forced to now interact with the new service instance. If the latency of service migration is an issue, CLOUDDEP can be configured to keep hot-spare instances of the service. These are running but unconfigured instances that have not yet received an identity. In the event of a service failure, CLOUDDEP can directly inject the identity into a hot-spare instance and create a more seamless failover.

C. Elastic Scalability

The third example combines the load balancing and the failover scenarios to enable elastic scalability of a remote service. In the event of a service over-utilization, CLOUDDEP can start a new instance of the service and bind some of the clients to the new service (see ❸ in Figure 2). A new identity is forked by CLOUDDEP from an existing configuration and can then develop independently from the original sibling. Scaling back to a smaller number of service instances can be achieved by applying the failover strategy: moving all clients of a service to a different existing instance and then dropping the service instance.

V. PROTOTYPE AND EXPERIMENTAL EVALUATION

We have implemented a prototype of CLOUDDEP that replicates configuration and manages failover for OSGi services in a cloud setup. The prototype uses R-OSGi as a distribution provider and cooperates with the protocol and the proxy handler to detect failures or service outages. The detection of failure is done through either a simple timeout or an exception caused by the termination of a connection. Since we induce failures in the experiments through terminating the connection, the latter detection is triggered. CLOUDDEP can furthermore either proactively start idle and unconfigured redundant instances of the service on a backup machine or reactively start a new instance of the service on demand. In either case, the identity of the failed service is injected into the new instance and the existing binding between the client and the no longer available service is transparently altered to point to the new instance. The prototype blocks the failed service request and replays the request to the new service instance as soon as it becomes available. We therefore assume a fail-stop model in our setup.

	Microreboot	Migration
avg. inv. time (T)	$7.23\mu s \pm 1.06\mu s$	$1.22ms \pm 0.15ms$
avg. T during failover	$266.29\mu s \pm 34.90\mu s$	$4.01ms \pm 0.66ms$
relative overhead	3581%	229%

Table I
SERVICE INVOCATION LATENCIES OF THE TREND SERVICE

For the experiment, we use a trend service that updates itself by periodically polling the most frequently used #hashtags from Twitter. The service is clustered by geographic regions, each instance of the service is configured to handle a specific region that is set through its configuration. Since the service only handles the ten most relevant results, it can afford to use the Configuration Admin interface to persist its internal state and does not rely on, e.g., an external blob storage. In a realistic setup, the service would process the results and, for instance, enrich the data with context links retrieved from Google Search or Wikipedia.

We use a single instance of the service and a single client that polls the service every second to receive the latest trends. After one minute, we take the service down and let CLOUDDEP handle the failure based on a strategy that we have pre-configured.

In the first experiment, we have both the client and the service running in the same Java Virtual Machine and on the same OSGi framework. Our failure recovery strategy resembles a microreboot of the service, i.e., we stop the running service and restart it. In the second experiment, we run client and service in different cloud instances and on different Java Virtual Machines and additionally keep a hot-spare VM for the failover. When the original service fails, we instantiate a new service instance on the hot-spare that becomes enabled when CLOUDDEP injects the configuration (i.e., geographic location and previously determined trends) into the instance.

Table I summarizes the measured average service invocation time during normal operation and during the failover. In the case of the microreboot, the invocation time during failover increases by a factor of 35 during failover. The reason for this high increase is that in OSGi clients get a direct reference to the service object and thereby a regular service invocation does not differ from any other method call in terms of latency. Enterprise Java Beans, in contrast, are called through reflection. Furthermore, a bundle restart is potentially more expensive than a microreboot of a single Enterprise Bean. However, the overall latency during failover is still in the range of microseconds, which is very likely acceptable for the large majority of applications.

In the distributed case the baseline of an average regular

service invocation is much higher and is in our setup in the order of milliseconds. However, the failover increases the invocation time by only a factor of three. In practice, this factor will vary with the amount of configuration data that needs to be migrated to restore the service identity and the network used but more complex services are likely to also have higher baseline invocation times. In both the local and the distributed case, only a single service invocation is temporarily blocked and experiences a higher latency. Further invocations continue to operate at normal latency.

The results illustrate the viability of our approach for a large class of applications. Microreboots are an option for recovery from failures, especially when the service is already accessed from a different host. In this case, the latency of the microreboot is within the noise of a remote service invocation. When entire nodes fail or become unreachable, migration of services to different machines are also feasible and do not affect the client significantly. We have, however, assumed that the cloud instance for failover is already instantiated. In our setup, we used an Eucalyptus cluster that requires in average 85 seconds to boot up a new instance to the point where we can interact with it. This latency is unlikely to be acceptable for cloud services but is a common problem of IaaS solutions.

VI. Conclusion

We expect dependability to become an increasing problem for cloud services. Our premise is that traditional ways of structuring such systems provide insufficient flexibility and cannot deal with failures or elasticity requirements as they occur in cloud setups. We have therefore proposed a modular, loosely-coupled approach for building cloud services and have illustrated the concrete challenges using the example of OSGi, a widely used standard for modularity for the Java language. The result of this effort is CloudDep, a dependability service for OSGi that takes care of countering service failures by migrating the identity of the service to a new machine. The experimental results of our prototype implementation show that migration to an existing node is feasible and the impact on the service invocation time is acceptable for most applications.

References

[1] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, 2004.

[2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, Apr. 2010.

[3] "Cloud Computing Incidents Database." [Online]. Available: http://wiki.cloudcommunity.org/wiki/CloudComputing:Incidents_Database

[4] K. Joshi, G. Bunker, F. Jahanian, A. van Moorsel, and J. Weinman, "Dependability in the cloud: Challenges and opportunities," in *Dependable Systems Networks, 2009. DSN '09. IEEE/IFIP International Conference on*, 29 2009-july 2 2009, pp. 103 –104.

[5] H. Stockinger, A. Samar, K. Holtman, B. Allcock, I. Foster, and B. Tierney, "File and object replication in data grids," *Cluster Computing*, vol. 5, no. 3, pp. 305–314, 2002.

[6] J. Abawajy, "Fault-tolerant scheduling policy for grid computing systems," in *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*. IEEE, 2004, pp. 238–244.

[7] OSGi Alliance, *OSGi Service Platform, Core Specification Release 4, Version 4.2*, 2009.

[8] B. Meyer, *Object-Oriented Software Construction*, 1st ed. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1988.

[9] C. Hewitt, P. Bishop, and R. Steiger, "A universal modular ACTOR formalism for artificial intelligence," in *Proceedings of the 3rd international joint conference on Artificial intelligence*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1973, pp. 235–245.

[10] J. Liedtke, "On micro-kernel construction," in *Proceedings of the fifteenth ACM symposium on Operating systems principles*, ser. SOSP '95. New York, NY, USA: ACM, 1995, pp. 237–250.

[11] G. Candea, S. Kawamoto, Y. Fujiki, G. Friedman, and A. Fox, "Microreboot - a technique for cheap recovery," in *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, ser. OSDI'04, 2004.

[12] P. Kriens, "How osgi changed my life," *ACM Queue*, vol. 6, no. 1, Jan. 2008.

[13] C. Matthews and Y. Coady, "Virtualized recomposition: Cloudy or clear?" *Software Engineering Challenges of Cloud Computing, ICSE Workshop on*, vol. 0, pp. 38–43, 2009.

[14] J. S. Rellermeyer, M. Duller, and G. Alonso, "Engineering the cloud from software modules," in *ICSE-Cloud 09: Proceedings of the Workshop on Software Engineering Challenges in Cloud Computing (in conjunction with ICSE 2009)*, Vancouver, Canada, 2009.

[15] J. S. Rellermeyer, G. Alonso, and T. Roscoe, "R-osgi: Distributed applications through software modularization," in *Middleware 07: 8th ACM/IFIP/USENIX International Middleware Conference*, Newport Beach, CA, USA, 2007.

[16] J. S. Rellermeyer, G. Alonso, and T. Roscoe, "Building, deploying, and monitoring distributed applications with eclipse and r-osgi," in *ETX 07: Fifth Eclipse Technology eXchange (ETX) Workshop (in conjunction with OOPSLA 2007)*, Montreal, Canada, 2007.

[17] OSGi Alliance, *OSGi Service Platform, Service Compendium Release 4, Version 4.2*, 2009.

[18] Apache Foundation, "Apache zookeeper," http://zookeeper.apache.org, 2008.

[19] OSGi Alliance, *OSGi Service Platform Enterprise Specifications, Release 4, Version 4.2*, 2010.

Appendix Rellermeyer 04

2012 IEEE/IFIP 42nd International Conference ...

2012 IEEE/IFIP 42ND INTERNATIONAL CONFERENCE ON DEPENDABLE SYSTEMS AND NETWORKS WORKSHOPS (DSN-W) JUNE 25-28, 2012 /

Corporate Authors:	International Conference on Dependable Systems and Networks Boston, Mass. (Workshops.), IEEE Computer Society. Fault-Tolerant Computing Technical Committee, IEEE XPlore Conference Proceedings Online, IFIP Working Group 10.4 on Dependable Computing and Fault Tolerance
Format:	Online, Conference Proceeding, Book
Language:	English
Published:	[Piscataway, NJ] : IEEE Computer Society, c2012.
Subjects:	Application software > Congresses. Fault-tolerant computing > Congresses.
Access:	Online version
Tags:	No Tags, Be the first to tag this record! Add Tag

Holdings Description Comments Similar Items Staff View

MARC

LEADER			00000cam a2200000Ia 4500
001			1477437
005			20140129112545.0
006			m o d
007			cr cn|---|||||
008			120905s2012 njua ob 100 0 eng d
020			**|a** 1467322644
020			**|a** 1467322652
020			**|a** 9781467322645
020			**|a** 9781467322652
035			**|a** (OCoLC)827775858
035			**|a** (OCoLC)ocn827775858
040			**|a** W2U **|b** eng **|c** W2U **|d** OCLCO **|d** IEEXO
049			**|a** PVUM
050		4	**|a** QA76.5
111	2		**|a** International Conference on Dependable Systems and Networks **|n** (42nd : **|d** 2012 : **|c** Boston, Mass.). **|e** Workshops.
245	1	0	**|a** 2012 IEEE/IFIP 42nd International Conference on Dependable Systems and Networks Workshops (DSN-W) **|h** [electronic resource] : **|b** June 25-28, 2012 / **|c** sponsored by IFIP WG 10.4 on Dependable Computing and Fault Tolerance, IEEE CS Technical Committee on Dependable Computing and Fault Tolerance , IEEE Computer Society.

SIMILAR ITEMS

The 42nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012)
Published: (2012)

Dependable Systems and Networks Workshop (DSN-W), 2013 43rd Annual IEEE/IFIP Conference on date 24-27 June 2013.
Published: (2013)

Proceedings of the 2009 IEEE/IFIP International Conference on Dependable Systems and Networks (DSN) June 29-July 2, Estoril, Lisbon, Portugal.
Published: (2009)

Proceedings of the 2010 IEEE/IFIP International Conference on Dependable Systems and Networks (DSN) 28 June-01 July, 2010, Chicago, IL /
Published: (2010)

2011 IEEE/IFIP 41st International Conference on Dependable Systems and Networks Workshops (DSN-W) June 27-30, 2011, Hong Kong /
Published: (2011)

246	1		|i Available from some providers with title: |a Dependable Systems and Networks Workshops (DSN-W), 2012 IEEE/IFIP 42nd International Conference on
246	3	0	|a 42nd International Conference on Dependable Systems and Networks Workshops
246	3	0	|a Forty-second International Conference on Dependable Systems and Networks Workshops
246	3	0	|a IEEE/IFIP 42nd International Conference on Dependable Systems and Networks Workshops
246	3	3	|a DSN-W 2012
260			|a [Piscataway, NJ] : |b IEEE Computer Society, |c c2012.
300			|a 1 online resource.
500			|a Title from conference web page, viewed September 5, 2012.
504			|a Includes bibliographic references.
506			|a Electronic access restricted to Villanova patrons.
598			|a 29-JAN-14
650		0	|a Application software |v Congresses.
650		0	|a Fault-tolerant computing |v Congresses.
710	2		|a IEEE Computer Society. |b Fault-Tolerant Computing Technical Committee.
710	2		|a IEEE XPlore Conference Proceedings Online.
710	2		|a IFIP Working Group 10.4 on Dependable Computing and Fault Tolerance.
856	4	0	|z Online version |u http://ezproxy.villanova.edu/login?URL=http://ieeexplore.ieee.org/servlet/opac?punumber=6255871 |t 0
994			|a 92 |b PVU
999	f	f	|i 716c09c6-19f1-5a74-9119-f6c29fae64ed |s eec1fe43-f468-5532-989e-3f24c5c998ed |t 0
952	f	f	|p Default |a Villanova University |b Villanova PA |c Falvey Memorial Library |d World Wide Web |t 0 |e QA76.5 |h Library of Congress classification |i Electronic Books |n 0

COLLECTIONS
Tools
Collection Development
Distinctive Collections
Highlighted Collections

RESEARCH SERVICES
Ask a Librarian
Guides and Tools
Teaching and Learning
Digital Research and Publishing Support

USING THE LIBRARY
Access
Spaces
Technology
Borrow and Request
Learning Commons
Services For...

ABOUT
Overview
Library Organization
Events
Visit
Contact Us
News
Library Initiatives
Give to the Library
Jobs @ Falvey

Support Our Library

VILLANOVA UNIVERSITY 800 Lancaster Ave., Villanova, PA 19085 610.519.4500 Contact

DIRECTIONS PRIVACY & SECURITY DIVERSITY HIGHER EDUCATION ACT MY NOVA DIRECTORY WORK AT VILLANOVA

Back to Top Ask Us: Live Chat

Appendix Rellermeyer 05

```
001     9931821030210145l
leader  02986cam a2200529 a 4500
005     20240131213016.0
006     m     o  d
007     cr cn|---|||||
008     120905s2012    njua    ob    100 0 eng d
050     #4$aQA76.5
040     ##$aW2U $beng $epn $cW2U $dOCLCO $dIEEXO $dOCLCF $dOCLCO $dOCLCQ $dOCL $dOCLCO $dOCLCQ $dIAS $dLND $dCEF $dOCLCQ $dLVT $dOCLCO $dOCLCQ
020     ##$a9781467322645
020     ##$a1467322644
020     ##$a9781467322652
020     ##$a1467322652
049     ##$aMAIN
111     2#$aInternational Conference on Dependable Systems and Networks $n(42nd : $d2012 : $cBoston, Mass.). $eWorkshops.
260     ##$a[Piscataway, NJ] : $bIEEE Computer Society, $c©2012.
245     10$a2012 IEEE/IFIP 42nd International Conference on Dependable Systems and Networks Workshops (DSN-W) : $bJune 25-28, 2012 / $csponsored by IFIP WG 10.4 on Dependable Computing and Fault Tolerance,
246     30$aForty-second International Conference on Dependable Systems and Networks Workshops
246     30$a42nd International Conference on Dependable Systems and Networks Workshops
246     30$aIEEE/IFIP 42nd International Conference on Dependable Systems and Networks Workshops
246     3#$aDSN-W 2012
246     1#$iAvailable from some providers with title: $aDependable Systems and Networks Workshops (DSN-W), 2012 IEEE/IFIP 42nd International Conference on
300     ##$a1 online resource
336     ##$atext $btxt $2rdacontent
337     ##$acomputer $bc $2rdamedia
338     ##$aonline resource $bcr $2rdacarrier
500     ##$aTitle from conference web page, viewed September 5, 2012.
504     ##$aIncludes bibliographical references.
650     #0$aApplication software $vCongresses.
650     #0$aFault-tolerant computing $vCongresses.
650     #6$aLogiciels d'application $vCongrès.
650     #6$aTolérance aux fautes (Informatique) $vCongrès.
650     #7$aApplication software. $2fast $0(OCoLC)fst00811706
650     #7$aFault-tolerant computing. $2fast $0(OCoLC)fst00921988
655     #7$aConference papers and proceedings. $2fast $0(OCoLC)fst01423772
710     2#$aIFIP Working Group 10.4 on Dependable Computing and Fault Tolerance.
710     2#$aIEEE Computer Society. $bFault-Tolerant Computing Technical Committee.
856     40$3IEEE Xplore $uhttp://ieeexplore.ieee.org/servlet/opac?punumber=6255871
856     40$3IEEE Xplore $uhttp://0-ieeexplore.ieee.org.emu.londonmet.ac.uk/servlet/opac?punumber=6255871
856     40$3IEEE Xplore $uhttps://proxy.library.carleton.ca/login?url=https://ieeexplore.ieee.org/servlet/opac?punumber=6255871
856     40$uhttp://liverpool.idm.oclc.org/login?url=https://ieeexplore.ieee.org/servlet/opac?punumber=6255871 $z<img src="/screens/gifs/go4.gif" alt="Go button" border="0" width="21" height="21" hspace="7"
994     ##$a92 $bOCACL
035     ##$a(OCoLC)827775858
035     ##$a(OCoLC)ocn827775858
```

TOP
SEND TO
VIEW ONLINE
ITEM DETAILS
LINKS

CONFERENCE PROCEEDING

2012 IEEE/IFIP 42nd International Conference on Dependable Systems and Networks Workshops (DSN-W) : June 25-28, 2012

International Conference on Dependable Systems and Networks (42nd : 2012 : Boston, Mass.). Workshops.

©2012: Piscataway, NJ : IEEE Computer Society

Online access

Send to

E-MAIL | PERMALINK | CITATION | PRINT | ENDNOTE | EASYBIB | EXPORT RIS | BIBTEX | EXPORT TO EXCEL

https://orbiscascade-washington.primo.exlibrisgroup.com/permalink/01ALLIANCE_UW/1juclfo/alma9916177452070145 2

COPY THE PERMALINK TO CLIPBOARD

View Online

Full text availability

IEEE Xplore conference proceedings [SHOW LICENSE]

Access for all on campus; login required from off-campus.

Details about who can access online resources

Item Details

Title	2012 IEEE/IFIP 42nd International Conference on Dependable Systems and Networks Workshops (DSN-W) : June 25-28, 2012
Author	International Conference on Dependable Systems and Networks (42nd : 2012 : Boston, Mass.). Workshops. IFIP Working Group 10.4 on Dependable Computing and Fault Tolerance. IEEE Computer Society. Fault-Tolerant Computing Technical Committee.
Attribution	sponsored by IFIP WG 10.4 on Dependable Computing and Fault Tolerance, IEEE CS Technical Committee on Dependable Computing and Fault Tolerance, IEEE Computer Society
Subjects	Application software -- Congresses Fault-tolerant computing -- Congresses Application software Fault-tolerant computing
Genre	Conference papers and proceedings
Language	English
Format	1 online resource
Alternate Title	Available from some providers with title: Dependable Systems and Networks Workshops (DSN-W), 2012 IEEE/IFIP 42nd International Conference on
Description	Title from conference web page, viewed September 5, 2012. Includes bibliographical references.
Publisher	Piscataway, NJ : IEEE Computer Society
Publication Date	©2012
Identifier	ISBN: 9781467322645 ISBN: 1467322644 ISBN: 9781467322652 ISBN: 1467322652


OCLC Number 827775858
Alma IZ Record Number 99161774520701452
Alma NZ Record Number 99318210302101451
Links
Staff View

Appendix Rellermeyer 06

RESEARCH-ARTICLE

Component-based scalability for cloud applications

Authors: Steffen Kächele, Franz J. Hauck Authors Info & Claims

CloudDP '13: Proceedings of the 3rd International Workshop on Cloud Data and Platforms • Pages 19 - 24
https://doi.org/10.1145/2460756.2460760

Published: 14 April 2013 Publication History Check for updates

19 484 Get Access

Abstract

Cloud computing enables access to an almost unlimited amount of resources combined with usage-based accounting. However, due to their design a lot of applications are not able to exploit the elasticity provided by the cloud. In this paper, we introduce several mechanisms that allow exploitation of the component structure of applications in order to scale them in a cloud computing cluster. We present our OSGi-inspired component framework COSCA that automatically manages elastic deployment of component-based applications. It isolates components of different applications and hides distribution using a virtualized and distributed OSGi-like framework. We present the results of several experiments which show that scalability of component-based applications benefits from such a platform. Moreover, we show how lightweight and agile component-based scale-out is. Our approach eases the usage of cloud resources and scalability for component-based applications.

References

[1] Apex: Salesforce on-demand programming language and framework. http://developer.force.com/. Google Scholar

[2] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. *Comm. ACM*, 51(1):107--113, 2008. Digital Library Google Scholar

[3] E. Gabriel et al. Open MPI: Goals, concept, and design of a next generation MPI implementation. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, volume 3241 of *LNCS*, pages 97--104. Springer, 2004. Google Scholar

Show all references

Cited By

View all

Frederiksen N, L. Møller E, Tuxen J, E. O'Neill S and Boesen M. (2023). Rapid Scaling of a Danish Public Health System Under COVID-19. 36th Bled eConference – Digital Economy and Society: The Balancing Act for Digital Innovation in Times of Instability: June 25 – 28, 2023, Bled, Slovenia, Conference Proceedings. 10.18690/um.fov.6.2023.46. (723-730). Online publication date: 12-Dec-2023.

https://doi.org/10.18690/um.fov.6.2023.46

Fourati M, Marzouk S and Jmaiel M. (2023). Towards Microservices-Aware Autoscaling: A Review. 2023 IEEE Symposium on Computers and

Communications (ISCC). 10.1109/ISCC58397.2023.10218213. (1080-1083). Online publication date: 9-Jul-2023.

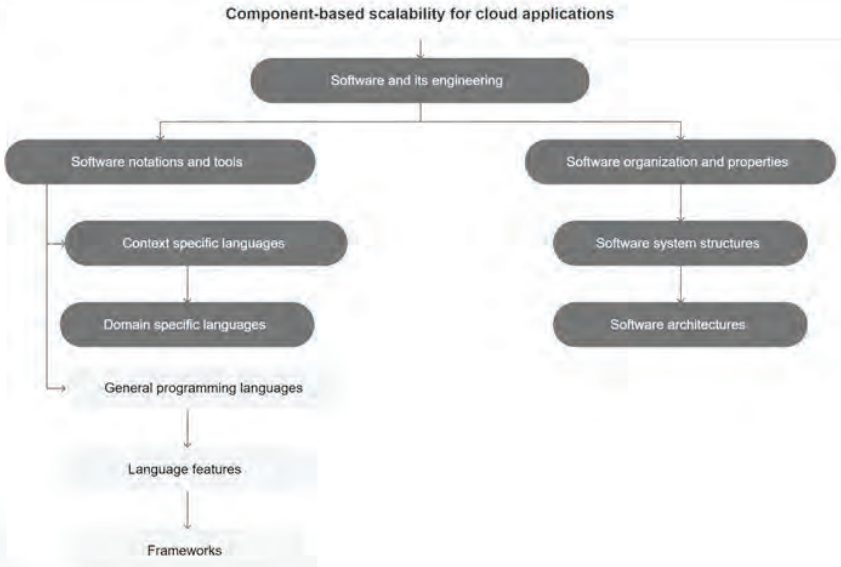
https://doi.org/10.1109/ISCC58397.2023.10218213

Gao Y, Zhao Y and Guo J. (2023). A Service Component Migration Method to maintain VM Performance. 2023 5th International Conference on Electronic Engineering and Informatics (EEI). 10.1109/EEI59236.2023.10212886. (170-180). Online publication date: 30-Jun-2023.

https://doi.org/10.1109/EEI59236.2023.10212886

Show More Cited By

Index Terms



Recommendations

Scalability, Elasticity, and Efficiency in Cloud Computing: a Systematic Literature Review of Definitions and Metrics

QoSA '15: Proceedings of the 11th International ACM SIGSOFT Conference on Quality of Software Architectures

Context: In cloud computing, there is a multitude of definitions and metrics for scalability, elasticity, and efficiency. However, stakeholders have little guidance for choosing fitting definitions and metrics for these quality properties, thus leading ...

Read More

DevOps patterns to scale web applications using cloud services

SPLASH '13: Proceedings of the 2013 companion publication for conference on Systems, programming, & applications: software for humanity

Scaling a web applications can be easy for simple CRUD software running when you use Platform as a Service Clouds (PaaS). But if you need to deploy a complex software, with many components and a lot users, you will need have a mix of cloud services in ...

Read More

A survey on elasticity management in PaaS systems

Elasticity is a goal of cloud computing. An elastic system should manage in an autonomic way its resources, being adaptive to dynamic workloads, allocating additional resources when workload is increased and deallocating resources when workload ...

Read More

Comments

DL Comment Policy

Comments should be relevant to the contents of this article, (sign in required).

Got it

0 Comments

+ Share

Best Newest Oldest

Nothing in this discussion yet.

Privacy Do Not Sell My Data

View Table Of Contents

The ACM Digital Library is published by the Association for Computing Machinery. Copyright © 2024 ACM, Inc.
of Usage | Privacy Policy | Code of Ethics



Component-based Scalability for Cloud Applications

Steffen Kächele, Franz J. Hauck
Institute of Distributed Systems, University of Ulm, Germany
{steffen.kaechele,franz.hauck}@uni-ulm.de

Abstract

Cloud computing enables access to an almost unlimited amount of resources combined with usage-based accounting. However, due to their design a lot of applications are not able to exploit the elasticity provided by the cloud. In this paper, we introduce several mechanisms that allow exploitation of the component structure of applications in order to scale them in a cloud computing cluster. We present our OSGi-inspired component framework COSCA that automatically manages elastic deployment of component-based applications. It isolates components of different applications and hides distribution using a virtualized and distributed OSGi-like framework. We present the results of several experiments which show that scalability of component-based applications benefits from such a platform. Moreover, we show how lightweight and agile component-based scale-out is. Our approach eases the usage of cloud resources and scalability for component-based applications.

Categories and Subject Descriptors D.2.11 [*Software Engineering*]: Software Architectures—Domain-specific architectures; D.3.3 [*Programming Languages*]: Language Constructs and Features—Frameworks

Keywords PaaS, cloud computing, scalability, elasticity, component, OSGi

1. Introduction

Cloud computing increases the flexibility of companies as they can move away from on-premise hosting to large scale hosting providers. Tenants can use a huge amount of resources with no need for a large initial investment. Another big advantage of cloud computing is elasticity. One problem, however, is to deploy and scale applications so that they can benefit from the sheer unlimited resources. In early cloud computing systems (i.e. IaaS), the resource allocation is manual. Tenants have to explicitly add resources, whereas the resource allocation in these systems is usually very coarse-grained (i.e. completely virtual machines). Thus, tenants have to split applications into pieces, deploy them onto virtual machine images, develop automatic resource allocation and do many administrative tasks such as application reconfiguration for seamless scaling, which is a complex task. In the past, a lot of new programming paradigms were developed towards scalability support for applications [1, 2]. New languages or even new programming models (e.g. Map Reduce) were designed. However, they fundamentally change the way applications are built and do not address the huge amount of existing code. They need a complete redesign of existing applications and finally do not match all applications.

In this paper, we argue that the component structure of well-designed software can be exploited to provide scalability in cloud computing systems. We propose a PaaS system on which component-based applications could be uploaded, are analyzed with respect to the component structure and used for distribution. It fully automates resource provisioning and deployment. We show how to monitor interaction between software modules, thus being able to decide when the ideal time is for scaling in or out. Our framework manages the distribution between cluster nodes and uses service interaction on node edges. As a consequence, resource allocation is implicitly provided. As scalability does not depend on large deployment units such as virtual machines, it is quite fine-grained and thereby very agile. Thus, it becomes possible to adapt applications in real-time to changes in load, infrastructure and objective function. Our prototype uses Java and a component model equivalent to OSGi [13].

Section 2 of this paper summarizes and discusses issues with respect to scalability. In Section 3, we introduce the OSGi programming model. Section 4 sketches the conceptual background of our component-based approach that is implemented and evaluated in Section 5. Before we conclude, we present related work in Section 6.

2. Scalability using a component structure

A major feature of cloud computing is *elasticity*. Applications that are compatible with the cloud model can make use of nearly unlimited resources. Elasticity is typically achieved

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
CloudDP '13, April 14, 2013, Prague, Czech Republic.
Copyright © 2013 ACM 978-1-4503-2075-7…$15.00

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0205

by scaling applications code. In general, scalability can be vertical or horizontal. *Vertical scalability* is addressed by increasing the power of nodes whereas *horizontal scalability* uses more nodes for the same job. The latter is mainly used in cloud computing.

One way to provide horizontal scalability is to replicate application code onto multiple nodes so as to distribute workload to available instances. Both the unit of replication and of distribution typically depend on the deployment unit. A IaaS deployment, e.g., can provide scalability by replicating virtual machines. Distributing and replicating a large amount of data, however, creates a lot of overhead and the way the platform can respond to workload peaks is rough.

The approach presented in this paper proposes a more fine-grained and thus agile structure by distributing *components*. Components are self-contained pieces of software. They are generally considered to be larger units of composition than objects [12]. In well-designed software, these components have high cohesion internally and low coupling to the outside. They encapsulate a dedicated functionality accessible via a special interface and are thus ideal parts for distribution. Distributing application on component level has many advantages. Using components, a platform can replicate exactly those parts of the application that form a bottleneck by introducing a minimum of overhead. On the other hand, when components use a lot of static resources (e.g. memory), the platform can keep the amount of instances low and improve its utilization by redirecting requests from many consumers.

3. OSGi

OSGi [13] is a common industry standard to modularize applications. It allows installation, updating, and uninstallation of software components at runtime without the need to restart the entire platform. Thus, it perfectly matches long running server applications that should remain reconfigurable.

The OSGi platform consists of multiple layers that provide different abstractions of interaction between components. The *module layer* is the lowest logical layer of OSGi. It defines the basis of the module concept. In OSGi, application code is parted into modules called *bundles*. They contain a manifest file describing bundle properties such as bundle name and version. Initially, bundles are completely isolated from each other. However, code sharing on the basis of Java packages is possible between bundles. For this purpose, the bundle manifest can specify which code is to be imported from other bundles and which of its own code is accessible for other bundles. OSGi framework implementations realize the isolation of bundles by assigning each bundle a separate Java class loader. It connects class loaders according to the imports in the respective bundle manifest.

The *service layer* realizes a services-oriented and thus loosely-coupled system. The major entity of this layer is the *service registry*. It allows bundles to register services under a certain interface. Other bundles can look up registered services in the registry and use them to implement their required functionality. Bundles are allowed to register and unregister services at any time. For this purpose, OSGi provides an event concept to notify service users about changes.

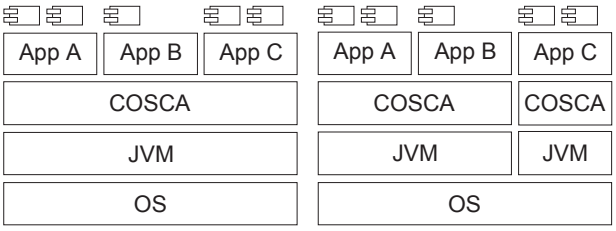
The *distributed OSGi specification* extends the OSGi model. It allows services running in another container to be accessed. The container can even run on another node. Remote services can be accessed as a usual service via the service registry. However, this does not cover scalability and cases with one consumer and multiple providers. When using a service proposed in the distributed OSGi specification all registered services appear as individual entities in the registry and thus it remains the task of the application to manage scalability and choose the references for each request. Yet, it provides no support for automatic scale-out.

4. COSCA

COSCA is a Java-based PaaS platform that uses an OSGi-inspired programming model [8]. It is designed to bring OSGi functionality to large scale systems.

COSCA aims at hosting multiple OSGi-based applications on shared third-party hardware. For deployment and management, it introduces the concept of applications that are composed of multiple bundles (i.e. components). COSCA applications can be directly deployed onto the platform. Being aware of this relationship, bundles that belong to an application are treated as a single entity. Yet, deploying multiple applications on a single node requires separation. For that purpose, the COSCA platform provides isolated OSGi-like environments for each application. We call these environments *virtualized frameworks*. Virtualized frameworks are very lightweight and aware of the mapping of bundles to applications (i.e. they only wire bundles of the same application). As multiple frameworks can run in the same JVM, creating a virtualized framework generates a very low footprint. COSCA can thus host multiple applications on a single host without using an additional IaaS layer that causes additional overhead. To separate applications, the platform uses different mechanisms on both module and service layers. On the module layer, it uses mechanisms of classloaders and security managers to separate applications. For isolation, the framework ensures that classloaders of different applications can only access their own specific code base and are not connected during wiring. Thus, bundles can only import packages that they export in an appropriate framework. Furthermore, we use security-manager features to prohibit access to Java functionality that may endanger isolation, and to limit access to dedicated resources (e.g. only parts of the file system). On the service layer, each application has its own service registry. Thus, when an application looks up a service, it will only see those it has registered before. Similarly, framework, bundle, and service events are only

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0206



(a) One instance (b) Multiple instances

Figure 1. Deployment and Architecture

published to the application of the corresponding tenant. An optional runtime layer [8] allows for additional separation, such as separating threads, file system, and network.

Fig. 1 summarizes the architecture. One option is (a) to run COSCA as a single entity on a computing node. It provides a central Java-based platform to deploy multiple applications in this case. When an application requires higher isolation (e.g. to prioritize application execution on OS level), it is (b) also possible to separate an application by spawning another dedicated JVM on this node. In this case, we have multiple instances of the platform running on one node. The COSCA frameworks with multiple nodes form a distributed PaaS system. Applications may have bundles that are deployed on multiple nodes. This allows us to address load balancing and scalability.

4.1 Component-based scalability

The OSGi service platform concept provides an elegant way of deploying component-based applications. However, current frameworks lack functionality to exploit scalability in the cloud. This includes platform support for distribution, automatic load balancing, and a mechanism that decides when to scale. It also requires automatic deployment support when the platform scales out.

For that purpose, we introduce a number of services for scalability support. In the following, we describe services that can be used to distribute applications among multiple nodes and to transparently access them using remote services. We describe strategies on how to handle services that exist on the local as well as on distributed nodes.

Distributed Service Registry Our distributed service registry is the central unit that manages and eases distributed deployment of applications. It supports adding the capacity of an additional node to an application by seamlessly extending a virtual framework among multiple nodes. It is aware of redundant service instances in case of replication. When applications get deployed on different nodes (e.g. during scale-out), our distributed service registry connects the virtual frameworks by synchronizing the services that are available for an OSGi application. Whenever a service gets registered, the registry automatically distributes the service by exporting it via a distribution provider (i.e. remote service specification) and publishes it within an application-specific distributed registry. Thus, the platform can handle applications that exceed the resources of one physical node, e.g., when the node does not have enough memory. Another example is an application that has high workload in some components. Replicating these components to an additional node is one solution in this case.

Platform support for elasticity In order to provide platform support for elasticity, we introduce the concept of *virtual services* that addresses two different issues. The first issue affects scaling support. To distribute applications among different nodes, we use the capabilities of the remote admin service that allows access of services on a remote host via a URL using an OSGi-compliant RPC. These services, however, appear as dedicated units in the service registry. In this case, the application has to cope with scalability which is complicated, error-prone and is not applicable to legacy applications. The second issue concerns the statically designed references to services. When a service consumer looks up a service in an OSGi framework, it will get a direct reference to the service implementation. As a consequence, the platform is no longer able to balance requests to another instance or to transparently migrate components within the cluster.

For supporting scalability and migration, virtual services decouple service consumers from service providers. Virtual services are automatically created by our distributed service registry and are a facade to real services. Our platform thus decouples the logical service reference from the service instance. Decoupling provides flexible mapping between the service reference and the actual service. Initially, in a non-replicated application, virtual services point to local services. When scaling, they can add further service instances to the pool of services (see Fig. 2). For scaling support, they support strategies such as a random strategy, a weighted random strategy (incorporating the workload of nodes), or even a strategy with stickiness to address state.

Monitoring and automatic deployment When deploying component-based applications, one challenge is to detect and locate overload situations. Whenever a node gets overloaded it becomes noticeable due to typical operating system parameters such as high CPU usage or high memory consumption. Whereas this is enough in virtual-machine–based deployments (e.g. IaaS), detection in a component-based application gets more complicated as there is no direct mapping of component utilization to operating system parameters. For a fine-grained automatic scale-out it is essential to locate bottleneck components. For that purpose, our platform includes a *monitoring service* observing service invocations that represent the interaction between components. To estimate utilization of a component, we can measure either the time an invocation takes or the resources such as CPU time a thread in a service call consumes. The platform thus knows which component consumes how much resources. When components consume a high percentage of local resources, it may be replicated. When resources of a

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0207

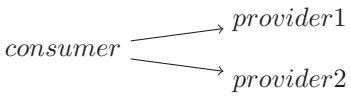


Figure 2. Virtual Services



Figure 3. Plot Application

node are exhausted, components are migrated. When monitoring resources, nested calls (i.e. when a service calls another service) are not easy to handle. Our monitoring service, therefore, uses a stack-based algorithm that stores CPU time consumed by nested invocations in the current stack frame. Thus the exact resource usage of the current component can be deduced by subtracting resources consumed outside, and overload situations can be detected.

To enable automatic scale-out and scale-in, we developed an internal REST-based API that allows frameworks to control the life-cycle of both applications and dedicated bundle instances of other frameworks (e.g., install, start, stop, update, and uninstall). An additional web interface allows users to manage and monitor applications.

4.2 Optimizations

Our platform has further optimizations to handle component distribution.

Coupling of components Components can have different types of coupling. Whereas loosely coupled components are ideal to scale, tightly coupled components may incur high communication load and dependability issues when distributing (e.g. when importing packages via the OSGi module layer). Depending on the coupling it may or may not be better to distribute components. For that purpose, our platform has a mechanism that detects the coupling between components. It mainly uses facts from OSGi wiring and investigates inter-component package imports. When there is high coupling (e.g. importing an implementation of a class), we treat these components as one unit which can only be scaled together. Conversely, the platform marks components with low coupling as scalable.

Addressing state When scaling services, state may become a challenge as it directly affects the possibilities of distribution and scaling. Whereas requests to stateless services can be arbitrarily distributed to available instances, multiple requests to stateful service have to be routed to the same service. On our platform this issue reflects on the component composition. The platform supports the issue on both deployment (i.e. component) and service (i.e. request) level. On the component level, the platform allows single components to be prohibited from scaling. Indeed, this limits the scalability of applications. In cases where it decreases performance, developers can further optimize their applications. They can exploit the component structure to separate state from computational components, e.g. by outsourcing them into a state management component. A particular extract is to use a service that manages state in the backend. Depending on the state, a—possibly replicated—database service can store state. A distributed coordination service (e.g. ZooKeeper) and the configuration admin service proposed by OSGi are examples of that. On the service level, the platform can use a sticky policy and wire single service consumers to distinct service providers so that they access the same state.

5. Implementation and Evaluation

In our implementation, we integrated the proposed services into our cloud platform COSCA [8]. We use R-OSGi as distribution provider and Zookeeper [7] as backend for our distributed service registry. Management services such as the backend and internal services for topology and infrastructure information run in an additional administrative application on a dedicated virtualized framework. As these services should stay highly available, we allow replication of the services to multiple nodes (in conjunction with Zookeeper).

For our evaluation, we use two nodes equipped with an Intel Core2Duo CPU with 4 GB of main memory connected by a one gigabit Ethernet switch.

5.1 Experiment A

In our first experiment, we deploy an application that visualizes data by plotting a given dataset into a PNG image. Fig. 3 shows the component structure of this application. A servlet-based frontend forwards requests to an images service that plots the image using the open source Java library JFreeChart.[1]

Methodology We use three nodes to generate a synthetic workload. Each node simulates eight clients that simultaneously access the web service. As a result, we simulate 24 clients that continuously request plots.

Experiment A1 We first run the application on one node. In this setup, we achieved 61 requests per second (Table 1). When stressing the application, the platform detects high workload in the image service and replicates the corresponding bundle to an additional node. After scale-out, we measured 118 requests per second (193.8 %). Although we use a simple random strategy that equally distributes the requests to available nodes, the performance almost doubles.

5.2 Experiment B

In our second experiment, we deploy an ordinary OSGi application that was originally not developed for scalability. It calculates ideal routes between given waypoints on a map.

[1] http://www.jfree.org/jfreechart/

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0208

	one node	two nodes
request/s	61 ± 2.35	118 ± 4.01
relative	100%	193.8%

Table 1. Experiment A1

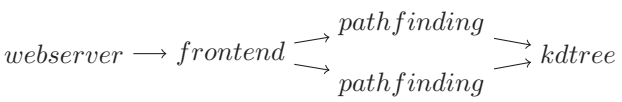


Figure 4. Navigation Application

Users can provide routes via a map on a website provided by the frontend bundle (see Fig. 4). The frontend uses a webserver to host the servlets. The path-finding service calculates the route using a contraction hierarchy algorithm [5]. It uses a KD-tree to map GPS coordinates to an internal data structure. This data structure is very efficient but consumes a lot of memory.

Methodology In our experiment, we use three nodes to generate a synthetic workload. Each node simulates 20 clients that simultaneously access the web service. As a result, we simulate 60 clients that continuously request routes.

Experiment B1 We first measured the speedup when this application is scaled-out due to high load (see Table 2). When running the application on a single node, we observe an overall throughput of 194 requests per second. After our monitoring service detects the high workload at the path-finding service, it scales it out on an additional node. As both the KD-tree and the frontend bundle cause relatively low workload, the platform will not replicate them. Especially not replicating the KD-tree saves a lot of memory. After automatic scaling, we measure a throughput of 312 routing requests per second (160.8 %). The difference (39 %) to linear scalability is twofold. On the one hand, each path-finding service has to communicate backwards to the KD-tree to resolve coordinates. This causes unnecessary overhead in a distributed environment. Modifying the path-finding service to start routing with already resolved coordinates may clearly improve performance. On the other hand, we also observe that while the first computing node is fully utilized the second is not. The reason is that we use a random strategy which uniformly distributes requests to available path finding services. As the first node has to manage webserver, frontend, KD-tree and one of the path-finding instances, it has more workload than the other node hosting just an instance of the path-finding service.

To address such a bottleneck, we may use a load-aware strategy and distribute requests with respect to the workload of nodes. Using such a strategy, the additional nodes would receive more workload in contrast to the node hosting the webserver, frontend bundle, and KD-tree. A deployment strategy could also improve utilization. For example, we could migrate the bundle with the routing service to another node so that the first node only hosts the webserver, frontend bundle, and KD-tree whereas the path finding service is hosted on the additional acquired nodes. Evaluation of these options is topic of further research.

	one node	two nodes
request/s	194 ± 2.68	312 ± 4.35
relative	100%	160.8%

Table 2. Experiment B1

	scale application	scale service
time(ms)	3212	64
relative	100%	2%

Table 3. Experiment B2

Experiment B2 In this experiment, we compare the time needed to scale-out individual components in contrast to a complete application. We measure how long it takes to start a new replica of the path-finding component compared to the time for starting a second routing application containing all components. The latter would be the default for scale-out in standard clouds. Table 3 shows the results. For our *path finding service*, we measure a startup time of 64 ms. In contrast, a complete application start up takes about 3200 ms, mainly caused by initializing the KD-tree. As a result, a component-based scale-out is about 50 times faster than replicating the complete application. Further benefit can be observed when comparing with a virtual machine replication in a IaaS setup.

6. Related work

In recent years, several new programming paradigms have been developed to support scalability and to map computation onto a cluster of nodes. One popular paradigm is Map Reduce [2] that Google introduced in 2008. It allows large jobs to be scale by splitting them into independent smaller ones to distribute them. Other examples are task processing systems from grid computing such as the Globus Toolkit [4] that uses web-service technologies to distribute grid jobs to a computational cluster. However, they only address computational jobs and do not match server applications. Message passing systems such as OpenMPI [3] are another mechanism to achieve scalability. In comparison to our work, they lack a mechanism for component composition and their programming model does not provide support for application deployment [10]. CORBA [18] introduces a component model [11] and proposes some parallel extensions [15], but the tight coupling between objects with respect to dependencies hinders flexibility and adaptability of applications. A^2-VM [17] and JESSICA [9] are JVMs to execute applications on multiple nodes. Instead of components, they use threads as a distribution unit combined with a distributed shared memory.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0209

There are already some commercial and open source platforms available that fully manage deployment. ConPaaS [14] is a PaaS platform that allows deployment of general server applications that consist of multiple parts. In contrast to COSCA, it uses a coarser granularity for composition (processes). As a consequence, to scale an application in ConPaaS it has to be compiled to multiple processes that may communicate through IPC mechanisms such as sockets. In COSCA, we use software components for distribution. We exploit the fact that in software engineering well-designed software already consists of multiple components that are loosely coupled. These components may reside in one process as long as they are running on one node. During scaling, we implicitly add IPC mechanisms between components. Google App Engine[2] provides a Python, Java and Go environment. Yet, it limits the programming model to web servlets and developers have to use the AppEngine's proprietary APIs that restrict application portability so as to run them exclusively in the Google cloud. Currently, it does not provide high-level support for composing applications from fine-grained components. OpenShift Flex[3] and Cloud Foundry[4] are open source PaaS systems. Yet, none of them supports component-based deployment and scaling.

The cloud provider Force.com proposes APEX [1], a new programming language to support scalability. New programming paradigms, however, require a fundamental redesign of existing applications. CLOUDdep [16] introduces a service to support easy replication and recovery from failures in OSGi cloud environments. For replication, they use a micro boot mechanism. In contrast to our work, they focus on a service that makes OSGi service references dependable.

7. Conclusion

Scaling of applications is not an easy task. In this paper, we propose an approach to exploit a component-based structure to scale applications within a cloud computing cluster. We introduce services that allow transparent resource allocation as well as distribution of components on multiple computing nodes. They monitor distinct components of an application and decide when to scale out and in. Our prototype uses Java and an OSGi model. It fully automates elastic cloud resource provisioning and does not require any manual setup. As there are OSGi framework implementations for a wide range of devices, cloud applications remain portable and thus it is easy to move them from on-premise hosting to the cloud and vice versa.

As cloud environments are often built up from commodity hardware, they tend to fail. In future work, we will evaluate how we can extend our methodology to such undependable environments in order to provide fault tolerance [6].

[2] http://appengine.google.com

[3] http://openshift.redhat.com/app/flex

[4] http://www.cloudfoundry.com

References

[1] Apex: Salesforce on-demand programming language and framework. http://developer.force.com/.

[2] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. *Comm. ACM*, 51(1):107–113, 2008.

[3] E. Gabriel et al. Open MPI: Goals, concept, and design of a next generation MPI implementation. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, volume 3241 of *LNCS*, pages 97–104. Springer, 2004.

[4] I. Foster. Globus toolkit ver. 4: Software for service-oriented systems. *J. of Comp. Sci. and Techn.*, 21:513–520, 2006.

[5] R. Geisberger, P. Sanders, D. Schultes, and D. Delling. Contraction hierarchies: faster and simpler hierarchical routing in road networks. In *Proc. of 7th Int. Conf. on Exp. Alg.*, WEA'08, pages 319–333. Springer, 2008.

[6] F. J. Hauck, S. Kächele, J. Domaschka, and C. Spann. The COSCA PaaS platform: on the way to flexible and dependable cloud computing. In *Proc. of the 1st Europ. Workshop on Dep. Cloud Comp.*, EWDCC '12, pages 1:1–1:2, NY, USA. ACM.

[7] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed. Zookeeper: wait-free coordination for internet-scale systems. In *Proc. of the 2010 Ann. USENIX Techn. Conf.*, USENIXATC'10, pages 11–11, CA, 2010.

[8] S. Kächele, J. Domaschka, and F. J. Hauck. COSCA: an easy-to-use component-based PaaS cloud system for common applications. In *Proc. of the First Int. Workshop on Cloud Comp. Platforms*, CloudCP '11, pages 4:1–4:6, NY. ACM.

[9] M. J. M. Ma, C.-L. Wang, and F. C. M. Lau. Jessica: Java-enabled single-system-image computing architecture. *J. Parallel Distrib. Comput.*, 60(10):1194–1222, 2000.

[10] Malawski et al. Component-based approach for programming and running scientific applications on grids and clouds. *Int. J. of High Perf. Comp. Appl.*, 26(3):275–295, 2012.

[11] R. Marvie and P. Merle. Corba component model. TR, 2002.

[12] P. Mougin and C. Barriolade. Web service, business objects and component models. Techn. Report, Oct. 2001.

[13] OSGi Alliance. OSGi service platform core spec. 4.3, 2011.

[14] G. Pierre and C. Stratan. ConPaaS: A platform for hosting elastic cloud applications. *Internet Comp., IEEE*, 16(5):88–92, 2012.

[15] C. Pérez, T. Priol, and A. Ribes. A parallel CORBA component model for numerical code coupling. In *Grid Comp.*, volume 2536 of *LNCS*, pages 88–99. Springer, 2002.

[16] J. Rellermeyer and S. Bagchi. Dependability as a cloud service - a modular approach. In *IEEE/IFIP 42nd Int. Conf. on Dep. Sys. and Netw. Workshops (DSN-W)*, pages 1–6, 2012.

[17] J. Simão, J. a. Lemos, and L. Veiga. A2-VM: a cooperative Java VM with support for resource-awareness and cluster-wide thread scheduling. In *Proc. of 2011th Confederated Int. Conf. on on the Move to Meaningful Internet Sys.*, OTM'11, pages 302–320. Springer, 2011.

[18] S. Vinoski. Corba: integrating diverse applications within distributed heterogeneous environments. *IEEE Comm. Mag.*, 35(2):46–55, Feb. 1997.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0210

Appendix Alves 01

LIBRARY OF CONGRESS CATALOG

LOGIN

LC Catalog Quick Search

SEARCH OPTIONS

Refine Your Search

BOOK

OSGi in depth

Full Record MARC Tags

000		01682cam a22003137a 4500
001		17180003
005		20120529132110.0
008		120224s2012 nyu 001 0 eng d
906	__	|a 7 |b cbc |c copycat |d 2 |e ncip |f 20 |g y-gencatlg
925	0_	|a acquire |b 2 shelf copies |x policy default
955	__	|b xn03 2012-02-24 z-processor to USPL |a xh00 2012-02-28 to USPL/STM |i xh58 2012-03-02 ; to Dewey
010	__	|a 2011284960
016	7_	|a 015734869 |2 Uk
020	__	|a 9781935182177
020	__	|a 193518217X
035	__	|a (OCoLC)ocn769218508
040	__	|a VYR |c VYR |d BTCTA |d UKMGB |d YDXCP |d CLE |d BWX |d BDX |d DLC
042	__	|a lccopycat
050	00	|a QA76.73.J38 |b A458 2012
100	1_	|a Alves, Alexandre de Castro.
245	10	|a OSGi in depth / |c Alexandre de Castro Alves.
260	__	|a Shelter Island, NY : |b Manning Publications Co., |c c2012.
300	__	|a xxii, 370 p. : |b ill. ; |c 24 cm.
500	__	|a Includes index.
520	2_	|a OSGi as a new platform for application development -- An OSGi framework primer -- The auction application: an OSGi case study -- In-depth look at bundles and services -- Configuring OSGi applications -- A world of events -- The persistence bundle -- Transactions and containers -- Blending OSGi and Java EE using JNDI -- Remote services and the cloud -- Launching OSGi using start levels -- Managing with JMX -- Putting it all together by extending Blueprint
650	_0	|a Java (Computer program language)
650	_0	|a Computer software |x Development.
856	42	|3 Contributor biographical information |u http://www.loc.gov/catdir/enhancements/fy1209/2011284960-b.html
856	42	|3 Publisher description |u http://www.loc.gov/catdir/enhancements/fy1209/2011284960-d.html

Request this Item LC Find It Item Availability



Book

PRINT RECORD

SAVE RECORD

EMAIL RECORD

CITE RECORD

MARCXML Record
MODS Record

Report Record Errors
Ask a Librarian
Feedback

Top

Item Availability

CALL NUMBER	QA76.73.J38 A458 2012 Copy 1
Request in	Jefferson or Adams Building Reading Rooms
Status	Not Charged

LIBRARY OF CONGRESS CATALOG

LOGIN

LC Catalog Quick Search

SEARCH OPTIONS

Refine Your Search 1 of 1

BOOK

OSGi in depth

Full Record MARC Tags

Personal name
Alves, Alexandre de Castro.

Main title
OSGi in depth / Alexandre de Castro Alves.

Published/Created
Shelter Island, NY : Manning Publications Co., c2012.

Request this Item LC Find It Item Availability


Book

PRINT RECORD
SAVE RECORD
EMAIL RECORD
CITE RECORD

MARCXML Record
MODS Record

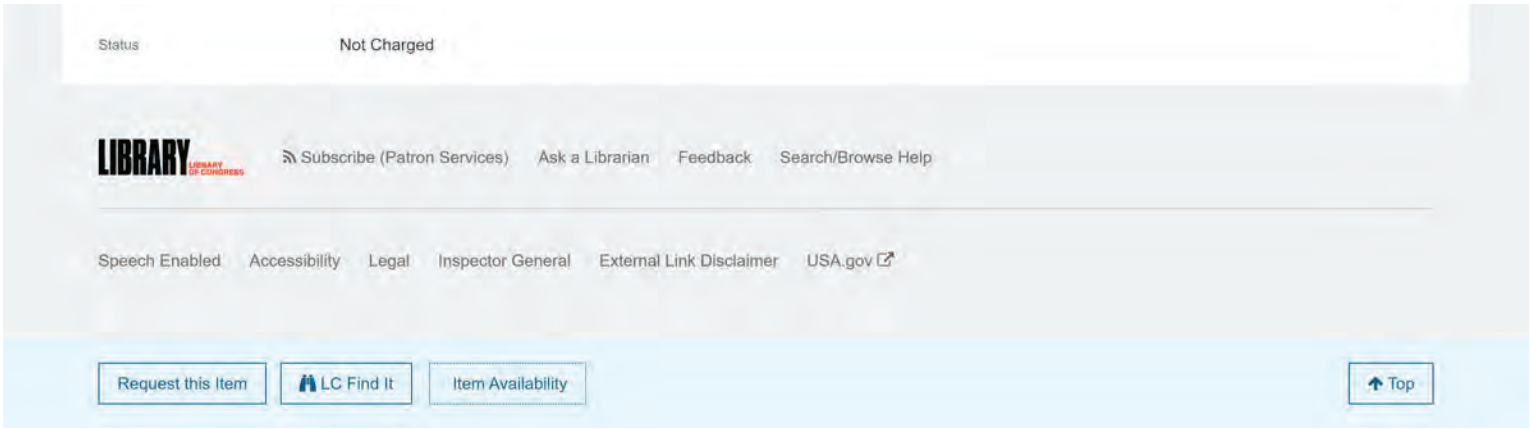
Report Record Errors
Ask a Librarian
Feedback

Top

Links	Contributor biographical information http://www.loc.gov/catdir/enhancements/fy1209/2011284960-b.html Publisher description http://www.loc.gov/catdir/enhancements/fy1209/2011284960-d.html
LCCN Permalink	https://lccn.loc.gov/2011284960
Description	xxii, 370 p. : ill. ; 24 cm.
ISBN	9781935182177 193518217X
LC classification	QA76.73.J38 A458 2012
Scope and content	OSGi as a new platform for application development -- An OSGi framework primer -- The auction application: an OSGi case study -- In-depth look at bundles and services -- Configuring OSGi applications -- A world of events -- The persistence bundle -- Transactions and containers -- Blending OSGi and Java EE using JNDI -- Remote services and the cloud -- Launching OSGi using start levels -- Managing with JMX -- Putting it all together by extending Blueprint
LC Subjects	Java (Computer program language) Computer software--Development.
Browse by shelf order	QA76.73.J38
Notes	Includes index.
LCCN	2011284960
National bib agency no.	015734869
Other system no.	(OCoLC)ocn769218508
Type of material	Book

Item Availability

CALL NUMBER	QA76.73.J38 A458 2012 Copy 1
Request in	Jefferson or Adams Building Reading Rooms


Status
Not Charged
LIBRARY
Subscribe (Patron Services)
Ask a Librarian
Feedback
Search/Browse Help
Speech Enabled
Accessibility
Legal
Inspector General
External Link Disclaimer
USA.gov
Request this Item
LC Find It
Item Availability
Top

Appendix Alves 02

```
LEADER 00000cam  2200385Ia 4500
001    769218508
008    111223s2012    nyu           001 0 eng d
020    9781935182177
020    193518217X
040    VYR|cVYR|dBTCTA|dUKMGB|dYDXCP|dCLE|dBWX|dBDX
049    RVEU
050  4 QA76.73.J38|bA43 2011
100 1  Alves, Alexandre de Castro
245 00 OSGi in depth /|cAlexandre de Castro Alves
260    Shelter Island, NY :|bManning Publications Co.,|cc2012
300    xxii, 370 p. :|bill. ;|c24 cm
500    Includes index
520 2  OSGi as a new platform for application development -- An
       OSGi framework primer -- The auction application: an OSGi
       case study -- In-depth look at bundles and services --
       Configuring OSGi applications -- A world of events -- The
       persistence bundle -- Transactions and containers --
       Blending OSGi and Java EE using JNDI -- Remote services
       and the cloud -- Launching OSGi using start levels --
       Managing with JMX -- Putting it all together by extending
       Blueprint
650  0 Java (Computer program language)
650  0 Computer software|xDevelopment
980    120302|e7500|f790042
981    GOBI
```

LOCATION	CALL NO.	STATUS
Wallace Library - Offsite Collections: Books	QA76.73.J38 A43 2011	AVAILABLE

RIT Libraries

Start Over | Request | Save Records | MARC Display | Return to Lists | Limit/Sort Search | Another Search

t:(osgi in depth) Entire Catalog System Sorted Search

Limit search to available items

Record 1 of 3

Previous Record Next Record



Author Alves, Alexandre de Castro
Title OSGi in depth / Alexandre de Castro Alves
Publisher Shelter Island, NY : Manning Publications Co., c2012

LOCATION	CALL NO.	STATUS
Wallace Library - Offsite Collections: Books	QA76.73.J38 A43 2011	AVAILABLE

Send to Phone

Permanent Link

ISSN/ISBN 9781935182177
193518217X
Descript xxii, 370 p. : ill. ; 24 cm
Notes Includes index
Summary OSGi as a new platform for application development -- An OSGi framework primer -- The auction application: an OSGi case study -- In-depth look at bundles and services -- Configuring OSGi applications -- A world of events -- The persistence bundle -- Transactions and containers -- Blending OSGi and Java EE using JNDI -- Remote services and the cloud -- Launching OSGi using start levels -- Managing with JMX -- Putting it all together by extending Blueprint
Subject Java (Computer program language)
Computer software -- Development

More Information

Browse Shelf Summary About The Author
You May Also Like

We may have ebooks and other electronic resources available. Look for the (e) below.

Browse Shelf



See Full Shelf

Summary

Summary

OSGi in Depth shows Java developers how to develop to the OSGi Service Platform Enterprise specification, an emerging Java-based technology for developing modular enterprise applications.

About the Technology

OSGi is a mature framework for developing modular Java applications. Because of its unique architecture, you can modify, add, remove, start, and stop parts of an application (read more)

About The Author



Alexandre de Castro Alves

Alexandre Alves is the architect for Oracle CEP, coauthor of the WS-BPEL 2.0 specification, and a member of the steering committee of EP-TS.

You May Also Like

Top Picks



See More Recommendations

Catalog enrichment powered by Syndetics Unbound

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0219

Appendix Alves 03

```
leader  02308cam  2200541Ia 4500
001     990020351440106761
005     20231211175756.0
008     111223s2012    nyu           001 0 eng d
010     ##$a  2011284960
016     7#$a015734869 $2Uk
019     ##$a694395422
020     ##$a9781935182177
020     ##$a193518217X
035     ##$a(MCM)002035144MIT01
035     ##$aNHCCYBP $b99947965741
035     ##$a(OCoLC)769218508
035     ##$aNHCCYBP $b99947965741
035     ##$a(OCoLC)694395422
040     ##$aVYR $cVYR $dBTCTA $dUKMGB $dYDXCP $dCLE $dBWX $dBDX $dDLC
049     ##$aMYGE
050     #4$aQA76.73.J38 $bA43 2011
099     ##$aQA76.73.J38 A43 2011
100     1#$aCastro Alves, Alexandre de.
245     10$aOSGi in depth / $cAlexandre de Castro Alves.
260     ##$aShelter Island, NY : $bManning Publications Co., $cc2012.
300     ##$axxii, 370 p. : $bill. ; $c24 cm.
336     ##$atext $btxt $2rdacontent
337     ##$aunmediated $bn $2rdamedia
338     ##$avolume $bnc $2rdacarrier
500     ##$aIncludes index.
520     2#$aOSGi as a new platform for application development -- An OSGi framework primer -- The auction application: an OSGi case study
650     #0$aJava (Computer program language)
650     #0$aComputer software $xDevelopment.
650     #7$aComputer software $xDevelopment. $2fast
650     #7$aJava (Computer program language) $2fast
776     1#$w(OCoLC)793913028
910     ##$aMARCIVEAUT
910     ##$aPromptCat $iPromptCat $d120418
910     ##$aMARCIVEAUT221103
910     ##$adepgy-035L210615 $d210615 $idepgy-035L
910     ##$aMARCIVEAUT231209
935     ##$l(OCoLC)793913028
946     ##$munpiggy-tang
949     #0$4IP $ae $bENG $cSTACK $o0 $p39080035521407 $x01 $hQA76.73.J38 A43 2011
959     #1$1MIT Access Only $an-mit $bNET $h**See URL(s) $o8 $x02
961     ##$aShelter Island, NY : Manning Publications Co.,   $a 2012  $a nyu
969     ##$aQA76.73.J38 $a694395422 $a769218508 $a694395422 $a  2011284960 $a9781935182177 $a193518217X
994     ##$a92 $bMYG
```



BOOK

OSGi in depth

Castro Alves, Alexandre de.

c2012

Available at Library Storage Annex Off Campus Collection (QA76.73.J38 A43 2011) >

Available Online >

Chapters of this book (19) >

Actions

PRINT CITATION EXPORT RIS EXPORT BIBTEX EXPORT TO EXCEL/CSV LINK EMAIL

Get it online

Online availability

O'Reilly Online Learning: Academic/Public Library Edition

For access, select "Institution not listed?" then enter your MIT email and authenticate through Touchstone.

Find/request items

MIT faculty, students, and staff: Please sign in to check if there are any request options. Sign in

BACK TO LOCATIONS

LOCATION ITEMS

Library Storage Annex

Available , Off Campus Collection ; QA76.73.J38 A43 2011

(1 copy, 1 available, 0 requests)

In library

More options

View record metadata - MARC format >

Details

Title	OSGi in depth
Creator	Castro Alves, Alexandre de. >
Subject	Java (Computer program language) > Computer software -- Development >
Description	OSGi as a new platform for application development -- An OSGi framework primer -- The auction application: an OSGi case study -- In-depth look at bundles and services -- Configuring OSGi applications -- A world of events -- The persistence bundle -- Transactions and containers -- Blending OSGi and Java EE using JNDI -- Remote services and the cloud -- Launching OSGi using start levels -- Managing with JMX -- Putting it all together by extending Blueprint

Related titles	Available in other form: Link to related record ›
Publisher	Shelter Island, NY : Manning Publications Co.
Creation Date	c2012
Format	xxii, 370 p. : ill. ; 24 cm.
Source	Library Catalog
Notes	Includes index.
Identifier	LC : 2011284960 ISBN : 9781935182177 ISBN : 193518217X OCLC : (OCoLC)769218508 OCLC : (OCoLC)694395422

Browse virtual shelf



John Zukowski's definitive guide to Swing for Java ...

1999



97 Things Every Java Programmer Should Know ...

2020



Mockito essentials : a practical guide to get you up and ...

2014



Proceedings of the 2001 Joint ACM-ISCOPE Conference on ...

2004



Java quick reference

1996

Appendix Alves 04

https://dl.acm.org/doi/abs/10.1145/2002259.2002261 September 30, 2024

ACM DIGITAL LIBRARY | Association for Computing Machinery

Browse | About | Sign in | Register

Journals | Magazines | Proceedings | Books | SIGs | Conferences | People

Search ACM Digital Library | Advanced Search

Home > Conferences > DEBS > Proceedings > DEBS '11 > A general extension system for event processing languages

RESEARCH-ARTICLE

A general extension system for event processing languages

Author: Alexandre De Castro Alves Authors Info & Claims

DEBS '11: Proceedings of the 5th ACM international conference on Distributed event-based system • Pages 1 - 10
https://doi.org/10.1145/2002259.2002261

Published: 11 July 2011 Publication History

Check for updates

2 296

Get Access

Abstract

Event processing languages (EPLs) for Complex Event Processing (CEP) systems are descriptive in nature. This allows the authoring of event processing applications at a higher level of abstraction. However, they are less suitable for dealing with low- level tasks, such as String manipulation, and other programming- in-the-small problems. In addition, current EPL implementations lack the richness of other programming language libraries (e.g. Java), which have been built over several years of usage. We present an architecture whereby providers can integrate existing programming languages into CQL, our EPL language of choice, leveraging the advantages of other languages, while still retaining the expressiveness and conciseness of EPL. We also demonstrate how the architecture avoids the awkwardness of previous integration solutions, which generally were focused only at the function-level (e.g. User-Defined Functions, Java call outs). Rather, the outlined solution is realized at the language level, through the extension of language implementation artifacts, such as type systems; thus resulting in a seamless, precise, and naturally blended environment for developing CEP applications. The architecture is based upon the concept of extension cartridges, which are modules that provide metadata describing the extensions. Finally, we show how these extension cartridges may be used in real-world CEP scenarios.

References

[1] Luckham, D. The Power of Events, An Introduction to Complex Event Processing in Distributed Enterprise Systems (2002).
Digital Library Google Scholar

[2] Arasu, A. and Babcock, B. and Babu, S. and Cieslewicz, J. and Datar, M. and Ito, K. and Motwani, R and Srivastava, U. and Widom, J. (2004) STREAM: The Stanford Data Stream Management System.
Google Scholar

[3] Schulte, R., Bradely, A.: A Gartner Reference Architecture for Event Processing Networks, ID G00162454, 2009.
Google Scholar

[4] Opher Etzion, EDA Conceptual Model (work in progress) -- IBM, 3rd EPTS Event Processing Symposium, Orlando, September 2007
Google Scholar

[5] Luckham, D. and Schulte, R. Event Processing Glossary -- Version 1.1 (2008).
Google Scholar

[6] A. Arasu, S. Babu, and J. Widom. The CQL Continuous Query Language: Semantic Foundations and Query Execution. Technical report, Stanford University, Oct. 2003. http://dbpubs.stanford.edu/pub/2003-67.
Google Scholar

[7] U. Srivastava and J. Widom. Flexible time management in data stream systems. In Proc. of the 23rd ACM SIGACT- SIGMOD-SIGART Symposium on Principles of Database Systems, June 2004.
Digital Library Google Scholar

[8] White, S., Alves, A., Rorke, D. WebLogic event server: a lightweight, modular application server for event processing. Proceedings of the second international conference on Distributed event-based systems (2008).
Digital Library Google Scholar

[9] Jain, N., Mishra, S., Srinivasan, A., Gehrke, J., Balakrishnan, H., Cetintemel, U., Cherniack, M., Tibbetts, R., Zdonik, S. Towards a streaming SQL standard. Proceedings of the VLDB Endowment (2008).
Digital Library Google Scholar

[10] Oracle CEP CQL Language Reference 11g Release 1. http://download.oracle.com/docs/cd/E12839_01/doc.1111/e12048/toc.htm
Google Scholar

[11] Oracle CEP IDE Developer's Guide for Eclipse. http://download.oracle.com/docs/cd/E12839_01/doc.1111/e14301/toc.htm
Google Scholar

[12] Esper for Java (version 3.3.0). http://esper.codehaus.org/esper/documentation/documentatio n.html
Google Scholar

[13] Coral8 Engine Documentation. http://www.aleri.com/developers/documents/coral8
Google Scholar

[14] Streambase and StreamSQL Documentation. http://streambase.com/developers-docs.htm
Google Scholar

[15] Yuri Leontiev, M. Tamer Zsu, and Duane Szafron. 2002. On type systems for object-oriented database programming languages. ACM Computing Surveys (CSUR) Surveys Volume 34 Issue 4, December 2002.
Digital Library Google Scholar

[16] James Gosling, Bill Joy, Guy L. Steele, The Java Language Specification, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 1996.
Digital Library Google Scholar

[17] François Bourdoncle, Stephan Merz, Type checking higher- order polymorphic multi-methods, Proceedings of the 24th ACM SIGPLAN-SIGACT symposium on Principles of programming languages, p.302--315, January 15-17, 1997, Paris, France{doi>10.1145/263699.263743}.
Digital Library Google Scholar

[18] Cardelli, L. 1989. Typeful programming. In Formal Description of Programming Concepts, E. J. Neuhold and M. Paul, Eds. IFIP State of the Art Reports Series. Springer- Verlag, New York. URL: http://www.luca.demon.co.uk/Bibliography.html.
Google Scholar

[19] OSGi Service Platform Version 4. http://www.osgi.org/Specifications/HomePage
Google Scholar

[20] Programming in the large and programming in the small. http://en.wikipedia.org/wiki/Programming_in_the_large_and _programming_in_the_small
Google Scholar

[21] Java Message Service. http://java.sun.com/products/jms/
Google Scholar

[22] Oracle CEP Data Cartridge Documentation,. http://download.oracle.com/docs/cd/E14571_01/apirefs.1111/e12048/datacartunder.htm#CCHCICHH
Google Scholar

[23] Oracle CEP Java Cartridge Documentation, http://download.oracle.com/docs/cd/E14571_01/apirefs.1111/e12048/datacartjava.htm#BGBEDIBC
Google Scholar

[24] Oracle CEP Spatial Cartridge Documentation, http://download.oracle.com/docs/cd/E14571_01/apirefs.1111/e12048/datacartspatial.htm#CHDBEAJA
Google Scholar

[25] Alves, A. 2011. OSGi in Depth, Manning
Digital Library Google Scholar

Show fewer references

Cited By

View all

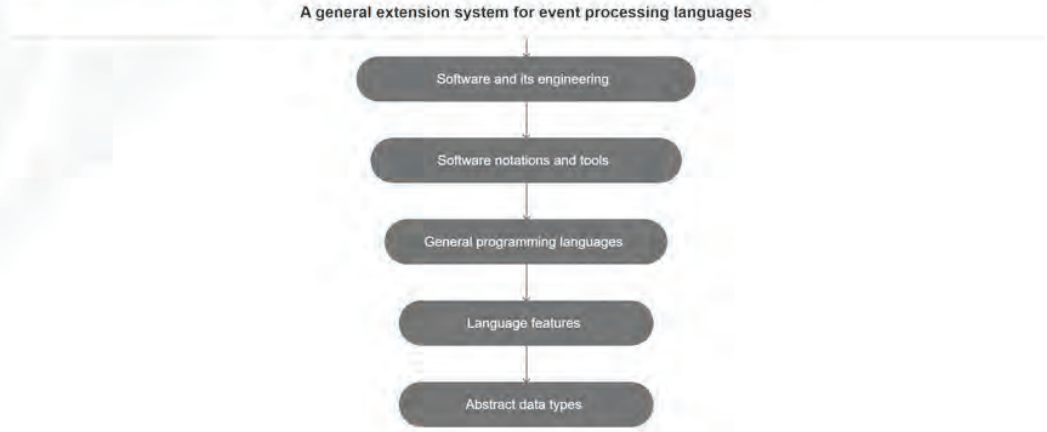
Rivera D, Monje F, Villagrá V, Vega-Barbas M, Larriva-Novo X and Berrocal J. (2019). Automatic Translation and Enforcement of Cybersecurity Policies Using A High-Level Definition Language. Entropy. 10.3390/e21121180. **21**:12. (1180). Online publication date: 30-Nov-2019.

https://doi.org/10.3390/e21121180

Nguyen F, Pitner T, Ivanović M and Budimac Z. (2012). Information system monitoring and notifications using complex event processing. Proceedings of the Fifth Balkan Conference in Informatics. 10.1145/2371316.2371358. (211-216). Online publication date: 16-Sep-2012.

https://dl.acm.org/doi/10.1145/2371316.2371358

Index Terms



Recommendations

Intrinsically-typed definitional interpreters for imperative languages

A definitional interpreter defines the semantics of an object language in terms of the (well-known) semantics of a host language, enabling understanding and validation of the semantics through execution. Combining a definitional interpreter with a ...

Read More

Characteristics of dynamic JVM languages

VMIL '13: Proceedings of the 7th ACM workshop on Virtual machines and intermediate languages

The Java Virtual Machine (JVM) has become an execution platform targeted by many programming languages. However, unlike with Java, a statically-typed language, the performance of the JVM and its Just-In-Time (JIT) compiler with dynamically-typed ...

Read More

Applying Optimizations for Dynamically-typed Languages to Java

ManLang 2017: Proceedings of the 14th International Conference on Managed Languages and Runtimes

While Java is a statically-typed language, some of its features make it behave like a dynamically-typed language at run time. This includes Java's boxing of primitive values as well as generics, which rely on type erasure....

Read More

Comments

DL Comment Policy

Comments should be relevant to the contents of this article, (sign in required).

Got it

0 Comments

Share

Best Newest Oldest

Nothing in this discussion yet

Privacy Do Not Sell My Data

View Table Of Contents

Categories

Journals
Magazines
Books
Proceedings
SIGs
Conferences
Collections
People

About

About ACM Digital Library
ACM Digital Library Board
Subscription Information
Author Guidelines
Using ACM Digital Library
All Holdings within the ACM Digital Library
ACM Computing Classification System
Accessibility Statement

Join

Join ACM
Join SIGs
Subscribe to Publications
Institutions and Libraries

Connect

Contact us via email
ACM on Facebook
ACM DL on X
ACM on Linkedin
Send Feedback
Submit a Bug Report

The ACM Digital Library is published by the Association for Computing Machinery. Copyright © 2024 ACM, Inc.
of Usage | Privacy Policy | Code of Ethics

A General Extension System for Event Processing Languages

Alexandre de Castro Alves
Oracle Corporation
300 Oracle Parkway, Redwood Shores, CA USA
+1-650-607-0878
alex.alves@oracle.com

ABSTRACT

Event processing languages (EPLs) for Complex Event Processing (CEP) systems are descriptive in nature. This allows the authoring of event processing applications at a higher level of abstraction. However, they are less suitable for dealing with low-level tasks, such as String manipulation, and other programming-in-the-small problems. In addition, current EPL implementations lack the richness of other programming language libraries (e.g. Java), which have been built over several years of usage. We present an architecture whereby providers can integrate existing programming languages into CQL, our EPL language of choice, leveraging the advantages of other languages, while still retaining the expressiveness and conciseness of EPL. We also demonstrate how the architecture avoids the awkwardness of previous integration solutions, which generally were focused only at the function-level (e.g. User-Defined Functions, Java call-outs). Rather, the outlined solution is realized at the language level, through the extension of language implementation artifacts, such as type systems; thus resulting in a seamless, precise, and naturally blended environment for developing CEP applications. The architecture is based upon the concept of extension cartridges, which are modules that provide metadata describing the extensions. Finally, we show how these extension cartridges may be used in real-world CEP scenarios.

Categories and Subject Descriptors

D.3.3 [**Programming Languages**]: Language Contructs and Features – *abstract data types --- SQL, CQL, Java, Spatial*

General Terms: Languages

Keywords

CEP, CQL, JAVA, PROGRAMMING-IN-THE-SMALL

1. INTRODUCTION

In section two, we start by presenting the scenario of an event-processing application that interacts with general-purpose libraries to perform miscellaneous (programming-in-the-small) tasks. This scenario helps us establish the requirements of the problem that we are addressing in this paper.

In section three, we present different known approaches that are being employed today for implementing the proposed scenario. We do so using Oracle CQL, however other EPL could have been used without loss of generality. We describe several disadvantages that these existing approaches have, and in light of these propose a new approach to solving the scenario, outlining its advantages.

In section four, we describe the architecture used to implement the new approach, which we denominate the *extensible language framework*.

In section five, we explain how the EPL and its extensions are used together seamlessly, following which in section six we describe the service provider interface for the extensible language framework itself.

In section seven, we provide the implementation of an industry use-case that makes use of the extensible language framework.

In section eight, we briefly elaborate on some of the details related to indexing and how they relate to the extensions.

Finally, in section nine, we conclude the paper, followed by discussing some of the related work in the area, and how it differs from the solution suggested in this paper.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
DEBS'11, July 11–15, 2011, New York, New York, USA.
Copyright 2011 ACM 978-1-4503-0423-8/11/07...$10.00.

2. SCENARIO

Consider a CEP application that is attempting to correlate sales of certain items to regional event news. For example, the result of a sport event may influence the sales of the winning team's merchandizes; or prolonged bad weather may increase the sales of weather-related items, such as umbrellas and raincoats. The goal of the CEP application is to determine which news events generate more sales, and which ones don't. Or, at a better instance, to quantify how much of a situation is needed to change the local spending behavior. For example, how many days of bad weather is needed to cause an increase on the average sales of umbrellas.

In this use-case, we can identify two sources of events, in other words, event streams. These are the *news* stream, and the *sales* stream. The news stream is unstructured in nature and is thus defined by a single *message* property of type *String*. The following event type defines the *sales* stream:

- item_description: String
- category: String
- location: String
- price: floating number

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0230

For our purpose, the *location* property specifies the city and state, comma-separated, where the item was sold. Following we have an instance of an event conforming to this type:

{item_description: “raincoat”, category: “cloth”, location: “San Francisco, CA”, price: 10.5}

As stated previously, our object is to correlate a news event to a sales event. To achieve this, the first step is to define our window of interest. Let’s set the window of time to 1 day using CQL:

```
SELECT * FROM
news [RANGE 24 HOURS],
sales [RANGE 24 HOURS]
WHERE …
```

Next, we need to define how the events will be correlated. To begin with, let’s naively use the location to correlate the events. However, this presents itself as a problem: the news event does not have a *location* property that can be used for *joining*. One solution is to search for a String in the news message that represents the two-letter acronym of a state, and use this sub-string as the join criteria. For the String search, we are only interested on whole-words that are uppercase. For example, “, CA” should match, whereas “caveat” or “CATEGORY” should not.

This type of String search is easily done with regular expressions. For example, the following pattern matches any two-letter uppercase String separated by a blank space or punctuations:

“[.,;][A-Z][A-Z][.,;]”

Regular expressions is a common feature of most programming languages. For example, the following code fragment does exactly this using the Java programming language [16]:

```
Matcher matcher =
Pattern.compile("[.,; ][A-Z][A-Z][.,; ]").matcher(message);
if (matcher.find()) {
  System.out.println("Location = "
    + message.substring(matcher.start() + 1,
    matcher.end() - 1));
}
```

Moreover, Java and other commonly used programming languages today in the market are feature rich; they support a wide spectrum of libraries, ranging from regular expression matchers, XML parsers, message transport and socket implementations, date formatting and handling utilities, String manipulation tools, etc. These libraries are ideal for *programming-in-the-small* [20], that is, code that do simple tasks that contribute towards the overall program’s objective, but is not part of the main program structure, design, or architecture.

Conversely, event-processing languages tend to focus on event processing-related verbs and subjects, such as the definition of the streaming window, as seen in Figure 1, and as such work at a higher level of abstraction, generally not being adequate for simple tasks and general-purpose code.

It is true that some EPL do have support for regular expressions, however this is generally in the form of a Boolean match, and wouldn’t work in this scenario where the actual matched String (e.g. “CA”) is needed.

As it can be concluded from the scenario, there is a need to be able to leverage these general-purpose libraries that are common today in the context of the EPLs. In the next section, we consider different ways to achieve this.

3. BLENDING LANGUAGES

Historically, the way used to invoke an external library is through the definition of user-defined functions (UDF).

3.1 User-Defined Functions

A UDF is a function defined by the application-writer that wraps some implementation code. In most cases, the implementation code must be authored using the platform’s native language. For example, the CEP platform evaluated in this paper is implemented in Java; therefore a UDF in CQL is likewise implemented in Java, as demonstrated next:

```
create function statematcher(msg char) return char
as language java instance “javaMatcher”
```

This statement defines a function with the adequate signature and associates the Java instance “javaMatcher” as its implementation, which must provide a *Object execute(Object[] args)* method:

```
public class JavaMatcher implements SingleElementFunction
{
 private Pattern pattern;

 public JavaMatcher()
 {
  pattern = Pattern.compile("[.,; ][A-Z][A-Z][.,; ]");
 }

 @Override
 public Object execute(Object[] args) throws UDFException
 {
  String ret = null;
  String msg = (String) args[0];

  Matcher matcher =
   pattern.matcher(msg);

  if (matcher.find()) {
   ret = msg.substring(matcher.start() + 1,
    matcher.end() - 1);
  }

  return ret;
 }
}
```

This approach has the following disadvantages:

1. Every library usage must be wrapped by a UDF signature. This is counter-productive. Consider that the String class alone has over 50 public methods, which would have to be wrapped.
2. UDFs are by their very nature solely defined as functions, lacking the flexibility and abstraction provided by object-oriented types, containing constructors, methods, and fields.
3. It lacks a seamless integration between the hosting event-processing language and the implementation language of the UDF. For example, no support for casting, etc.
4. UDF is implemented using the platform’s native programming language. For example, one would not be able to implement a UDF in C when running on a Java platform, without incurring the (productive) costs of

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0231

extending the platform itself (i.e. Java Native Interface - JNI).

An alternative approach to using UDF is to use the general-purpose library in the *action handler* of the EPL statement.

3.2 Action Handlers

The action handler of an EPL statement is the code *that is called back* when a *match* happens. This code is responsible for handling the matched event. For example, by sending the matched event to some external system through a messaging infrastructure.

In the CEP platform used, the action handler is implemented as Java objects outside of the declarative section that contains the specification of the CQL statements. In other words, one can visualize this CEP application as a graph of two nodes, the first node contains the CQL statements and defines the matching logic, the second node, which is downstream to the first node, contains the Java class that implements the action logic. The second node is trigged only when the first node finds a match. This graph of sorts is known as the event-processing network (EPN).

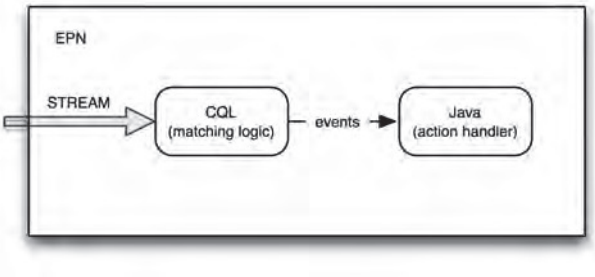


Figure 1. EPN containing Java code as an action handler

To implement our scenario using this approach, we would have to let the events pass-through and implement the matching logic in the Java code

```
SELECT * FROM news[RANGE 24 HOURS]
```

This is so because we would not be able to leverage the Java code directly in the WHERE clause. This is highly undesirable, as we would not be exploiting the declarative nature of the EPL.

For other EPLs, the action handler is coded as the right-hand-side in the EPL statement itself. In other words, the left-hand-side defines the match, and the right-hand-side defines the action. Nonetheless, in spite of the collocation of the matching and action logic, this approach is in essence similar to the previous approach employed where the action handler is part of a separate EPN node. This is so because there is still a clear delimitation between the matching and handler code, both in terms of the programming language used and the lifecycle of the entities involved. In other words, the action handles is still a separate code, implemented using a different language. This is demonstrated with the following example using a hypothetical event-processing language:

```
NEWS(message) AND SALES(location) :=
Matcher matcher =
  Pattern.compile("[.,; ][A-Z][A-Z][.,; ]").matcher(message);
if (matcher.find()) { … }
```

Although the action handler approach does not require a wrapping of the general-purpose libraries as the UDF approach does require, the action handler approach still yields several disadvantages, such as:

1. The EPL (i.e. matching logic) and the action handlers don't share context. This makes it hard to share parameters and results. For example, in our scenario, we would not be able to use the action handler logic directly within the WHERE clause of the CQL statement.
2. There is still no blending between the EPL and the action handler's programming languages. In other words, the user is clearly using two different languages, which entails on a less cohesive development environment.

What is needed is an approach that blends the languages, allowing the application writer to leverage both the declarative approach of the EPL together with the feature-rich libraries of other programming languages. In the next section, we propose a solution that addresses this.

3.3 Extensible Language Framework

Consider the following CQL statements that implements our scenario in question:

```
CREATE VIEW filtered_news(message, matcher) AS
SELECT message,
  Pattern.compile("[.,; ][A-Z][A-Z][.,; ]").matcher(message)
FROM news [RANGE 24 HOURS]

SELECT location, item_description, message
FROM filtered_news, sales[RANGE 24 HOURS]
WHERE matcher.find() = true AND
  news.message.substring(matcher.start() + 1,
  matcher.end() - 1) = sales.location
```

Let's investigate this in parts. We begin with the definition of a view that receives the news stream and generates a relation containing the last 24 hours of news messages and a matcher pattern object. The latter expression is the one that is subject of interest to us:

```
Pattern.compile("[.,; ][A-Z][A-Z][.,; ]").matcher(message)
```

This expression is invoking the static method *compile(String)* from the Java class *java.util.regex.Pattern*, following which it invokes the method *matcher(String)* on the object of the class *Pattern* that was returned from the *compile()* method. The return object from the method *matcher()* is stored as the attribute *matcher* of the generated relation represented by the view definition:

```
CREATE VIEW filtered_news(message, matcher)
```

Next, we use the *matcher* object to perform the join criteria between the *NEWS* stream and the *SALES* stream:

```
WHERE matcher.find() = true AND
  news.message.substring(matcher.start() + 1,
  matcher.end() - 1) = sales.location
```

In particular, we invoke several methods from the Java class *Matcher*, such as *find(), start(),* and *end().* Note how the methods *start()* and *end()* need to be invoked on the same object that is being invoked the method *find(),* otherwise they will raise an illegal state exception. Next, we invoke the method *String.substring(),* using the return from the previous methods as arguments after evaluating a simple mathematical expression (e.g. + 1). The result from the *substring*() method is used as the join criteria together with the sales location.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0232

Let's consider another example, where the user sends a message to an external system with the result. In this case, we are using the Java Message Service (JMS) [21] as the messaging infrastructure.

```
SELECT
  ApplicationTopicPublisher.getSingleton().
    publish(TextMessage(message))
…
```

In this case, we first invoke the constructor *TextMessage(String)* for the class *TextMessage*, passing to it the *message* attribute. Then we retrieve the singleton *TopicPublisher* object using the static method *ApplicationTopicPublisher.getSingleton()* and invoke its *publish()* method to send the message.

In the first example, we only used classes provided by the Java runtime environment, such as *String* and *Pattern*. In the second example, notice how we reference the class *ApplicationTopicPublisher*, which is being provided by the CEP application itself.

This set of examples show cases a tight integration between the EPL, which we denominate the *extensible language*, and the Java programming language, which we call the *extension language*. We shall call the overall approach here outlined as the *extensible language framework*.

All in all, the approach used in this section yields the following advantages:

1. No wrapping is needed to reference the general-purpose libraries. For example, the *Pattern* class is used directly.
2. There is a natural blending between the EPL and the language used to handle the general-purpose library. For example. the expression *"news.message.substring()"* nests the reference to a method (i.e. substring) of an attribute (i.e. message) of an stream (i.e. news).
3. Even though the examples restrict themselves to using Java as the extension language, the underlying architecture allows for any programming language to be used, and not just that of the hosting platform (as we shall see in the next section).
4. Types between the extensible and the extension languages are converted seamlessly, thus allowing the support for overloading, overriding and casting. In addition, one is able to use the extension language (e.g. Java) anywhere in the extensible language (e.g. EPL), including in the WHERE clause, thus possibly participating in a join criteria.

In the next section we explore the architecture used to implement this approach.

4. ARCHITECTURE

To be able to achieve a tight integration between the programming languages, they must share a common set of underlying concepts.

Upon close investigation of the examples provided in the previous section, one commonality that exists between both the extensible and extension language is that of attribute *types*. For example, the expression "matcher.find()" can be re-phrased as the invocation of an instance method on the complex type associated to the stream attribute *matcher*. Likewise, the expression "Pattern.compile(…)" is indeed the invocation of an static method on the complex type *Pattern*. Furthermore, the expression "matcher.start() + 1" can be implemented as the conversion of the extension simple type *INT* (from the Java programming language) into the equivalent extensible native type *INTEGER* present in CQL.

In other words, all extension types have some mapping into the extensible language. However, this must be done in such a way that the extensible language does not need to know about any of the implementation details of the extension, but rather only needs to know enough to *make use of it*. Otherwise we would be duplicating the extension language within the extensible language.

In addition, the extension languages are handled as plug-ins of the extensible language. That is, the extensible language provides a registry, which must be used by any extension language that wishes to participate in the framework. This is similar to any other plug-in framework, such as the one provided by the OSGi service platform [19].

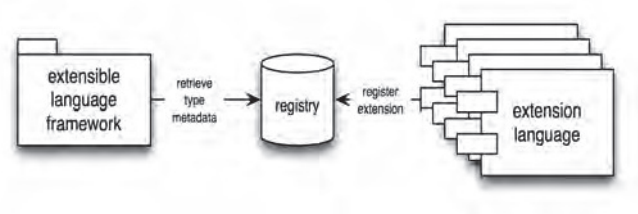


Figure 2 Plug-in architecture for extensible language framework

Let's look at the details of an example.

Consider the expression "Pattern.compile(…)". When the extensible language parses this expression, it breaks it into two identifiers, *Pattern* and *compile.* Next, it must find what kind of identifier the word "Pattern" represents. To do so, first it checks if there is any attribute named *Pattern* in any of the streams being used by the statement. If this check fails, the next check is to find if there is any type named *Pattern* in either the extensible language (e.g. CQL) or any of the plugged-in extension languages (e.g. Java). For this particular case, let's consider that the only extension language that has been plugged-in is the Java extension language. There is no type called *Pattern* in CQL, however the Java extension language does contain a type called *Pattern*, whose metadata is returned to the extensible language. Having retrieved the metadata for the identifier "Pattern" successfully, we can now move to the next identifier, the word "compile()". We know that "compile()" is a method invocation, because of the parentheses, likewise we know that it must be a member of the type represented by the previous identifier, because of the '.' operator. Hence, we check for the following conditions in the metadata returned for the identifier "Pattern":

- The metadata must be of a complex type (as opposed to a simple type).
- The metadata must contain a method whose name is "compile" and takes as parameter the type *String*.

If these conditions are met, the expression is found to be semantically correct. The final step is to handle the runtime invocation of an extension type. We do this by retrieving a function reference from the type's metadata for any of its members that are executable. For example, considering our previous case, the extensible language retrieves from the complex type *Pattern* a function reference that represents the execution of the method

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0233

compile(). This function reference is then used later during the runtime processing of events by the CQL runtime environment. In other words, we use the concept of a function as the abstraction for a runtime executable component within our extensible language framework. This plays well, as a function is indeed the lowest denominator of other executable components, such as methods and constructors. For example, a method can be converted into a function whose first argument is the 'this' pointer for the method's object instance. In addition, functions are passable of being optimized in the query plan generated for the EPL (e.g. CQL) statements.

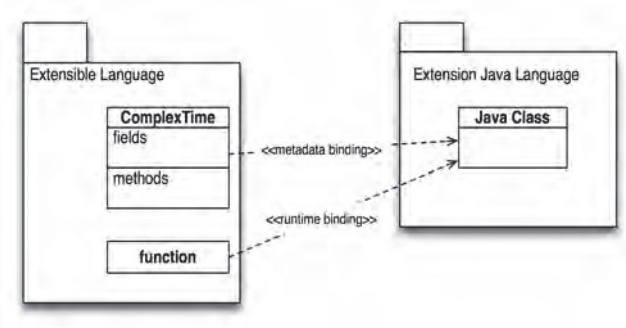


Figure 3 Extensible Language bindings to Java

Finally, it is noteworthy to highlight how the extensible language does not need to know any of the implementation details of the extension language. The extensible language framework only needs to know about the metadata of (complex) types, and how to generically invoke functions.

5. THE API

The extensible language framework described here has been implemented in Oracle CEP.

The implementation consists of changes to several modules of the CEP system, crossing several layers of the product, such as the following ones:

1. Parser module
2. Semantic Analyzer module
3. Registry module
4. Code Generation module
5. Runtime Code Executor module

However, from a user's perspective, the notable differences are modifications to the CQL language, or more precisely, to its grammar.

Specifically, the grammar is changed to support the following two concepts:

Complex Types

Links

These two important concepts are described next.

5.1 Complex Types

As it can be extrapolated from the Java based examples employed in the previous sections, the concept of an abstract type is a useful encapsulation mechanism for programming languages, one that is commonly materialized as *Classes* by object oriented languages. Therefore, to facilitate integration to popular languages, such as the Java programming language, we introduce support for complex types in CQL.

A complex type in CQL is a structured type made of the collection of other components. These components are called attributes (i.e. fields), and may be typed as either complex or simple. CQL supports the simple types one would generally expect, such as INT, BIGINT, BIGDECIMAL, FLOAT, DOUBLE, BYTE, STRING, BOOLEAN, as well as the perhaps less common (application) types of TIMESTAMP, INTERVAL, and XML.

The syntax for accessing a complex type's attribute is:

l-value '.' type-attribute-name

For example, consider the following expression:

SELECT S1.a1.f1 FROM S1

In this case, *S1.a1* is referencing to the attribute *a1* of the stream *S1*, which is of type complex. Next, *a1.f1* is referencing to the field *f1* of the object (l-value) represented by *a1*. As the identifier *S1* is optional in this query, the example can be re-arranged to:

SELECT a1.f1 FROM S1

Furthermore, a CQL complex type may also define methods and constructors for manipulating the objects of these types. The syntax for invoking methods and constructors is similar to that of most other object-oriented languages. Following up with the previous example, let's say that *S1.a1* is of type *java.lang.String*. In this case, the next example demonstrates how to invoke a method in the stream's attribute *a1*:

SELECT a1.f1.toString() FROM S1

Invoking a complex type's constructor creates a new stream attribute of the type, as in the following example:

SELECT java.lang.String("foo") as a1 FROM S1

That is, a new object of type *String* is created and associated as the value of the attribute "a1" of the output stream of the query.

Complex types can be used in any CQL clause that is able to handle stream attributes. In other words, complex types can be used not only as part of the *selection* list, but also in *where* predicates, *order by* lists, etc.

Next, we take a look at the concept of links.

5.2 Links

Types can be defined by metadata. For example, the metadata of the type *java.lang.String* says that the type is a class, containing the method *toString(),* amongst other methods and constructors.

Types are not the only construct that can be defined by metadata. For example, the native CQL function '*to_float(int): float'* is defined by a metadata that states that it takes a *int* argument, and returns a *float*. A function's metadata may even include the actual code to be executed when such function is invoked.

In the case of our extension framework, the metadata of the extensions need to be provided by the extension system itself and not by the native CEP implementation. For example, let's consider the Java class *String* used in the previous examples. When the Semantic Analyzer reaches the expression "f1.toString()", it needs to retrieve the type's metadata for the attribute *f1* and check if it contains the definition of a *toString()* method. In other words, the expression "f1.toString()" needs to be type checked appropriately.

However, who owns the definition of the Java type *String*? The native CEP implementation knows nothing of Java, or of any other

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0234

programming language, it only knows of the native CQL types and functions.

It is the role of links to point to the location of the metadata of the extensions. A link is defined as an identifier used in combination with the '@' character and placed as a suffix of an extension's name. For example, the following expression informs the CEP system (i.e. the extensible framework) that the extension type *String* lives in the Java system:

```
SELECT java.lang.String@java("foo") as a1 FROM S1
```

Other examples of expressions with links are:

```
my-extension-function@external-system("arguments")
contain@spatial(shape1, shape2, dist)
```

In summary, a link ties an extension type or function to its metadata providing system. This metadata is retrieved by the Registry module and used by the Semantic Analyzer module to type check the expressions. Following, the Code Generation module uses the metadata to generate the proper instructions to be executed at runtime by the Runtime Code Executor module. This is described in Figure 4.

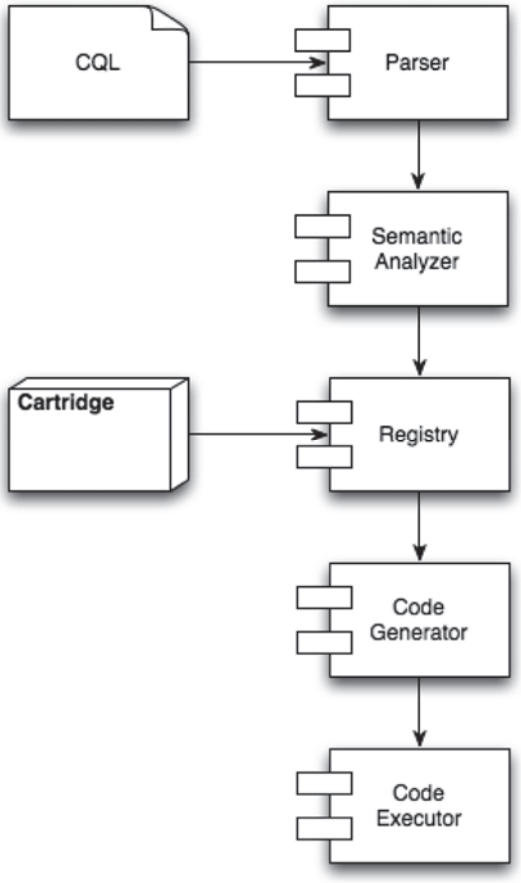


Figure 4 CEP layered modules and cartridge interaction

By default, if no link is specified, then the CEP system assumes that the Java extension module defines the extension. Therefore, the following two expressions are identical:

```
SELECT java.lang.String("foo") as a1 FROM S1
SELECT java.lang.String@java("foo") as a1 FROM S1
```

There is no need to specify the link if the l-value in question is already associated to a particular extension system. For example, in the expression "SELECT a1.toString() …", there is no need to specify the location of *toString()* metadata, as it is clear that it is a method defined by the complex type of the attribute *a1*. However, if the l-value does not provide this information, then the specification of the link is mandatory. This is the case of the invocation of a static method.

Having understood how a user makes use of CQL language extensions, our next task is to understand how vendors can plug-in new extension systems. This is the subject of the next section.

6. THE SERVICE PROVIDER INTERFACE

As we have seen previously, the CEP system allows extension types and functions to be used in its EPL. The extension systems that provide the metadata for the extensions types and functions are called *CEP cartridges*.

CEP cartridges are registered in a global Cartridge Registry. The key used in this registry is the link name. For example, the literal "java" is used as the key for the Java CEP cartridge. This means that when an extension is used with the link "@java", the CEP system expects the Java CEP cartridge to provide the needed metadata for the extension in question.

Vendors may register their own CEP cartridges. All cartridges must implement the following (pseudo) interface:

```
Cartridge
{
    TypeLocator getTypeLocator();
    FunctionMetadataLocator getFunctionMetadaLocator();
    IndexInfoLocator getIndexInfoLocator();
    //…
}
```

As evident, the role of the cartridge is to locate the proper metadata for its extensions.

The *TypeLocator* interface is defined as:

```
TypeLocator
{
    Type getType(String extensionName);
    // …
}
```

For example, to locate the definition of the extension type *java.lang.String*, the CEP system finds the cartridge for the implicit "java" link, and then invokes the method *TypeLocator.getType("java.lang.String")* of the Java cartridge.

The interface *Type* can be casted to *ComplexType*, which is defined as:

```
ComplexType
{
    MethodMetadata getMethod(String name,
            Type… params);
    AttributeMetadata getAttribute(String name);
    ConstructorMetadata getConstructor(String name,
    Type… params);
    //…
}
```

Next, let's drill down in the metadata of a method:

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0235

```
MethodMetadata
{
    Type[] getParameterTypes();
    Type getReturnType();
    Method getImplementation();
    //…
}
```

Finally, we need to define the implementation of the method itself:

```
Method
{
    Object invoke(Object target, Object [] args)
}
```

Although this may seem very similar to Java's reflection API, the reader should keep in mind that a CEP cartridge is working at a higher level of abstraction, therefore method metadata may be provided for non-Java types. For example, one may implement a Hadoop CEP cartridge, providing metadata for Hadoop's Bag (complex) type. In addition, a CEP cartridge also allows one to define non-Java concepts, such as function metadata, and as we shall see in following sections, extensible indexes.

At the time of writing, several CEP cartridges have been developed, some of which are provided by default in the Oracle CEP product:

Java CEP cartridge: allows a CEP developer to seamlessly make use of Java code in CQL statements. The cartridge not only converts native types to Java, but also Java to native types, therefore allowing one to switch back and forward between Java and CQL.

Spatial CEP cartridge: allows a CEP developer to make use of Spatial functions, such as contains, inside, etc; and Spatial types, such as Geometry, to implement location-enabled business intelligence solutions.

JDBC CEP cartridge: allows a CEP developer to make use of SQL functions being provided by remote RDBMs.

Next, let's look into more details at the Java CEP cartridge.

6.1 The Java cartridge

The Java cartridge provides us with a great level of flexibility in the CEP applications, therefore allowing us to fulfill our programming-in-the-small scenarios.

However, to make it fully usable, three additional issues must be dealt with. First and foremost, we need to define how the Java CEP cartridge should load the Java classes being referenced in the CQL statements.

In the examples so far we have been using Java classes defined by the runtime library, such as *String*. These do not present any problems, as they are always present in Java's parent (root) class-loader. However, how should we handle classes defined by the CEP applications themselves? In Oracle CEP, a developer can deploy multiple CEP applications into the system, each defining their own set of CQL queries and Java classes. The Java classes can be used at different locations at the CEP application's EPN (Event Processing Network) [3]. It is even plausible that different applications may define the same Java Class (e.g. MyClass).

Therefore, the Java CEP cartridge provides two configurable options for loading the Java classes referenced in CQL:

1. Application-scoped class-space: In this option, the Java cartridge only loads classes defined by the CEP application that owns the CQL statements.
2. Server-scoped class-space: In this option, the Java cartridge considers all classes that are made public by all libraries and applications of the running CEP system.

In case of conflicts, a resolution is reached using the OSGi service platform [25], which is the underlying application server infrastructure being used by Oracle CEP. Additional details around how OSGi is being used are outside of the scope of this paper.

The second issue is that of usability. Java classes generally have long names, such as *java.util.regex.Pattern*, which tend to decrease readability of the programs. This is minimized in Java with the use of package imports, however CQL has no such feature.

To allow us to have shorter CQL statements, such as "SELECT Pattern.compile(…)" instead of "SELECT java.util.regex.Pattern.compile(…)", the Java CEP cartridge introspects the MANIFEST.MF file of the CEP applications and considers all OSGi Import-Package headers. For example, let's say an application includes the follow header:

```
Import-Package: java.util.regex
```

Such application may omit the package name in the CQL query, therefore keeping it shorter: "SELECT Pattern.compile(…)". Again, the full details of OSGi wiring and its usage by the Java cartridge are outside of the scope of this paper.

Finally, it should be noted that the Java cartridge considers the full set of resolution rules as specified by the Java Specification Language (JSL), and therefore supports auto-boxing of the Java types. As an example, consider the following CQL expression:

```
attribute.methodA(10)
```

Where *attribute* is of type *mypackage.MyType*, which defines the following overloaded methods:

```
methodA(int)
methodA(Integer)
methodA(Object)
methodA(long)
```

As the literal 10 is of the primitive type *int*, the order of precedence is:

```
methodA(int)
methodA(long)
methodA(Integer)
methodA(Object)
```

In the next section, let's take a look at a real world scenario that makes use of the Spatial CEP cartridge.

7. APPLICATION EXAMPLE

Consider an application that wishes to provide specialized marketing for a mobile subscriber. The application verifies the location of the subscriber and if within a certain distance of a particular shop, it notifies the subscriber with a discount or advertising message (e.g. SMS). The role of CEP in this application is to check if the location of the subscriber is within the distance of any registered shop.

To accomplish this scenario, the CEP application deals with two sources:

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0236

1. *LocationStream*: stream of events describing the current location of the subscriber in terms of longitude (*lng*) and latitude (*lat*) coordinates and the customer id (*custId*).
2. *ShopRelation*: a table of registered shops. A shop has two attributes, its *id*, and a *geometry* object representing its location as a polygon.

We assume that some form of location tracking adapter (i.e. GPS) generates the events as defined by the *LocationStream*. Next, we define a CQL view that converts the longitude and latitude coordinates to a Spatial cartridge type:

```
<view id="CustomerPositionStream">
    SELECT createPoint@spatial(
                lng, lat) as point, custId
    FROM LocationStream
</view>
```

The Spatial cartridge defines a *createPoint()* extension function, which takes as argument the coordinates and returns the Spatial extension *Point* type.

Next, we simply join the *CustomerPositionStream* with the *ShopRelation* applying the right predicate and temporal window:

```
SELECT
    loc.custId, shop.id
FROM
    CustomerPositionStream[NOW] AS loc,
    ShopRelation AS shop
WHERE
    contain@spatial(shop.geometry,
            loc.point, 2.0d) = true
```

This query outputs the customer and shop id for all customers within 2 miles of all registered shops considering the latest location of a customer.

In the next section, we discuss one important aspect of the spatial cartridge, which is that of extensible indexes.

8. EXTENSIBLE INDEXING

So far we have primarily dealt with extension types and extension functions. Another very useful extension is that needed for indexing operators. For example, consider the case of the Spatial CEP cartridge, and the previous query where the spatial *contain* function is employed:

```
SELECT … WHERE  contain@spatial(geometry, point, d)
```

To improve the efficiency of the execution of the queries, it is common for indexing data structures to be used. However, the proper structured depends on the type of the operands and operators. For example, it would be advisable to hash functions when the operand is a String and the operation is *equals*.

In the case of the Spatial cartridge, the operands involved are generally that of geometry types, for which an appropriate indexing data structure is the R-Tree. However, the native CEP system does not provide such a structure, as it is not needed for its own native functions and operators. What we conclude is that we must also support the extension of the indexing data structures to be used by the appropriate extension functions. For example, the spatial *contain* function should index its relation operand with an R-Tree, essentially creating a minimum bounding rectangle (MBR) and improving the first pass filtering of checking if a point is within a geometry.

The full details of extensible indexing is being the scope of this paper, however it is here mentioned as it is an optional but yet very important aspect of CEP cartridge.

9. CONCLUSIONS

The extensible language framework allowed us to implement the scenario at hand in the most productive form, no additional implementation artifacts are needed, such as the creation of UDFs; furthermore there is a natural and efficient blending of languages.

The architecture employed to implement the extensible language framework provided a solution that is both flexible, as it allows different language bindings to co-exist, as well as scalable, as each language binding is optionally implemented in a separate physical plug-in or cartridge, which can be potentially distributed.

Three cartridges are described, namely the Java cartridge, the Spatial cartridge, and the JDBC cartridge. These cartridges allow a CEP developer to author complex applications, such a geotagging application, in a succinct and efficient manner. Furthermore, vendors may provide other CEP cartridges at will, thus validating the extensibility framework provided by the CEP cartridges infrastructure.

Further work is needed around the propagation of exceptions across the extension and the extensible languages.

10. RELATED WORK

In Esper [12], one can invoke Java methods directly from within its EPL, which is called EQL. This is done without the need of wrappings (e.g. UDFs). However, the integration between EQL and Java is still superficial, no constructors can be invoked, and there is no direct mapping between the native types and the Java types, hence features such as overloading and overriding are problematic. Furthermore, Java is the only supported extension language supported. If one desires to use a different programming language, then a different Esper product, implemented in a different platform, must be used.

Other CEP system implementations were investigated, and it was found that these tend to provide both mechanisms of UDFs (section 3.1) and action handlers (section 3.2). The UDFs are implemented using the platform's native language, generally being C++ and Java. The action handlers, which were sometimes called adapters, have both a co-located version using the platform's native language, as well as a remote version. The vendors provided SDKs in diverse languages (e.g. Python, Java) for the remote adapters.

11. ACKNOWLEDGMENTS

The Oracle CEP team.

12. REFERENCES

[1] Luckham, D. The Power of Events, An Introduction to Complex Event Processing in Distributed Enterprise Systems (2002).

[2] Arasu, A. and Babcock, B. and Babu, S. and Cieslewicz, J. and Datar, M. and Ito, K. and Motwani, R and Srivastava, U. and Widom, J. (2004) STREAM: The Stanford Data Stream Management System.

[3] Schulte, R., Bradely, A.: A Gartner Reference Architecture for Event Processing Networks, ID G00162454, 2009.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0237

[4] Opher Etzion, EDA Conceptual Model (work in progress) – IBM, 3rd EPTS Event Processing Symposium, Orlando, September 2007

[5] Luckham, D. and Schulte, R. Event Processing Glossary – Version 1.1 (2008).

[6] A. Arasu, S. Babu, and J. Widom. The CQL Continuous Query Language: Semantic Foundations and Query Execution. Technical report, Stanford University, Oct. 2003. http://dbpubs.stanford.edu/pub/2003- 67.

[7] U. Srivastava and J. Widom. Flexible time management in data stream systems. In Proc. of the 23rd ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 2004.

[8] White, S., Alves, A., Rorke, D. WebLogic event server: a lightweight, modular application server for event processing. Proceedings of the second international conference on Distributed event-based systems (2008).

[9] Jain, N., Mishra, S., Srinivasan, A., Gehrke, J., Balakrishnan, H., Cetintemel, U., Cherniack, M., Tibbetts, R., Zdonik, S. Towards a streaming SQL standard. Proceedings of the VLDB Endowment (2008).

[10] Oracle CEP CQL Language Reference 11g Release 1. http://download.oracle.com/docs/cd/E12839_01/doc.1111/e12048/toc.htm

[11] Oracle CEP IDE Developer's Guide for Eclipse. http://download.oracle.com/docs/cd/E12839_01/doc.1111/e14301/toc.htm

[12] Esper for Java (version 3.3.0). http://esper.codehaus.org/esper/documentation/documentation.html

[13] Coral8 Engine Documentation. http://www.aleri.com/developers/documents/coral8

[14] Streambase and StreamSQL Documentation. http://streambase.com/developers-docs.htm

[15] Yuri Leontiev, M. Tamer Zsu, and Duane Szafron. 2002. On type systems for object-oriented database programming languages. ACM Computing Surveys (CSUR) Surveys Volume 34 Issue 4, December 2002.

[16] James Gosling , Bill Joy , Guy L. Steele, The Java Language Specification, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 1996.

[17] François Bourdoncle , Stephan Merz, Type checking higher-order polymorphic multi-methods, Proceedings of the 24th ACM SIGPLAN-SIGACT symposium on Principles of programming languages, p.302-315, January 15-17, 1997, Paris, France [doi>10.1145/263699.263743].

[18] Cardelli, L. 1989. Typeful programming. In Formal Description of Programming Concepts, E. J. Neuhold and M. Paul, Eds. IFIP State of the Art Reports Series. Springer-Verlag, New York. URL: http://www.luca.demon.co.uk/Bibliography.html.

[19] OSGi Service Platform Version 4. http://www.osgi.org/Specifications/HomePage

[20] Programming in the large and programming in the small. http://en.wikipedia.org/wiki/Programming_in_the_large_and_programming_in_the_small

[21] Java Message Service. http://java.sun.com/products/jms/

[22] Oracle CEP Data Cartridge Documentation, . http://download.oracle.com/docs/cd/E14571_01/apirefs.1111/e12048/datacartunder.htm#CCHCICHH

[23] Oracle CEP Java Cartridge Documentation, http://download.oracle.com/docs/cd/E14571_01/apirefs.1111/e12048/datacartjava.htm#BGBEDIBC

[24] Oracle CEP Spatial Cartridge Documentation, http://download.oracle.com/docs/cd/E14571_01/apirefs.1111/e12048/datacartspatial.htm#CHDBEAJA

[25] Alves, A. 2011. OSGi in Depth, Manning

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0238

Appendix Alves 05



All proceedings of this conference

Smart Grid - Integration von Haushalten mittels Vernetzung und Energie-Community

Conference: VDE-Kongress 2012 - Intelligente Energieversorgung der Zukunft
11/05/2012 - 11/06/2012 at Stuttgart, Deutschland

Proceedings: **VDE-Kongress 2012**

Pages: **6** Language: **german** Typ: **PDF**

Personal VDE Members are entitled to a 10% discount on this title

15.00 € Paper

Add to Cart

Search Conference Papers

Overview with all Proceedings

Authors:
Trick, U.; Steinheimer, M.; Ruhrig, P. (Fachhochschule Frankfurt am Main, Kleiststraße 3, 60318 Frankfurt/M., Deutschland)
Tönjes, R.; Hölker, D. (Hochschule Osnabrück, Albrechtstraße 30, 49076 Osnabrück, Deutschland)

Abstract:
Den Problemen, die sich aus der zunehmend volatilen und immer mehr dezentralen Energieerzeugung sowie den heutigen Verteilnetzen als „Black Box" ergeben, wird hier durch die Einführung eines neuen Konzepts der Peer-to-Peervernetzten Energie-Community begegnet. Ausgehend von den Erfahrungen mit Kommunikationsmehrwertdiensten wird ein Service Management Framework eingeführt. Es ermöglicht zum einen personalisierte Dienste für das Smart Home und insbesondere das Energiemanagement inkl. der Einbindung von Multimediakommunikation. Zum anderen bietet es eine komfortable Vernetzung von Haushalten, Energielieferanten, Verteilnetzbetreibern und anderen mit den Zielen Kostenersparnis, Vermeidung von Netzausbau, Energieeinsparung, CO2-Reduzierung etc. Erste Simulationen bestätigen die Wirksamkeit dieses neuen und leistungsfähigen Ansatzes, wobei vor allem die Vorteile von Batteriespeichern in den Haushalten, aber auch die möglichen Beiträge von Demand Side Integration (Lastverschiebung) offensichtlich werden.

© VDE VERLAG GmbH, Berlin - Offenbach

Smart Grid - Integration von Haushalten mittels Vernetzung und Energie-Community

U. Trick[1], M. Steinheimer[1], P. Ruhrig[1], R. Tönjes[2] und D. Hölker[2]
[1] Fachhochschule Frankfurt am Main, Kleiststraße 3, 60318 Frankfurt/M., trick@e-technik.org
[2] Hochschule Osnabrück, Albrechtstraße 30, 49076 Osnabrück

Kurzfassung

Den Problemen, die sich aus der zunehmend volatilen und immer mehr dezentralen Energieerzeugung sowie den heutigen Verteilnetzen als „Black Box“ ergeben, wird hier durch die Einführung eines neuen Konzepts der Peer-to-Peer-vernetzten Energie-Community begegnet. Ausgehend von den Erfahrungen mit Kommunikationsmehrwertdiensten wird ein Service Management Framework eingeführt. Es ermöglicht zum einen personalisierte Dienste für das Smart Home und insbesondere das Energiemanagement inkl. der Einbindung von Multimediakommunikation. Zum anderen bietet es eine komfortable Vernetzung von Haushalten, Energielieferanten, Verteilnetzbetreibern und anderen mit den Zielen Kostenersparnis, Vermeidung von Netzausbau, Energieeinsparung, CO_2-Reduzierung etc. Erste Simulationen bestätigen die Wirksamkeit dieses neuen und leistungsfähigen Ansatzes, wobei vor allem die Vorteile von Batteriespeichern in den Haushalten, aber auch die möglichen Beiträge von Demand Side Integration (Lastverschiebung) offensichtlich werden.

Abstract

The problems arising from the increasingly volatile and decentralized power generation as well as the today's "black box" distribution networks are solved here by the introduction of a new concept of a peer-to-peer networked energy community. Based on the experience with communication value-added services a service management framework is introduced which allows on the one side personalized services for the smart home, especially the energy management, and includes the integration of multimedia communications. On the other side the concept provides a convenient networking of households, energy suppliers, distribution network operators and others with the objectives cost savings, avoidance of network expansion, energy saving, CO_2 reduction, etc. Initial simulations confirm the effectiveness of this new and powerful approach, most notably the benefits of battery storage in homes, but also the potential contributions of demand side integration (load shift).

1 Einführung

In den elektrischen Energieversorgungsnetzen der Zukunft geht die Tendenz hin zu stärker volatiler regenerativer Energieerzeugung mittels Windkraft- und Photovoltaikanlagen und weg von der zentralen hin zur dezentralen Erzeugung und Einspeisung, auch im bisher nicht überwachten Ortsnetz. Um trotzdem eine stabile und qualitativ gute Energieversorgung sicherstellen zu können, muss das Netz ausgebaut und vor allem durch Informations- und Kommunikationstechnik intelligent gemacht werden. Diese Entwicklung wird allgemein durch den Begriff „Smart Grid“ [1] umschrieben. Zur Lösung dieser Problematik werden verschiedene, sich zum Teil ergänzende Ansätze diskutiert. Erwähnt seien hier die Stichworte „Netzausbau“ [2; 3], „Virtuelle Kraftwerke“ [4; 5; 3] (vgl. auch Aktivitäten der Energielieferanten Lichtblick und Vattenfall), „Automatisierung der Verteilnetze“ [6; 3], „Zwangsabschaltung von Erzeugern und/oder Verbrauchern“ [7; 8], aber auch „Steuerung durch Preissignale“ [9]. Letzteres fällt unter die von der Bundesnetzagentur mit dem Begriff „Smart Market“ versehenen und präferierten Maßnahmen [2].

Trotz umfangreicher Forschungs- und Entwicklungsaktivitäten auf den Gebieten Smart Grid und Smart Home in den letzten Jahren wurden bei den diskutierten Lösungen bzw. veröffentlichten Forschungsergebnissen die nachfolgend skizzierten Lücken bzw. Mängel nicht behoben:

- Zwar werden dem Nutzer eines Smart Home Dienste für die Haushaltsautomatisierung und das Energiemanagement angeboten [10], allerdings wird die für den Nutzer in diesem Zusammenhang hilfreiche Unterstützung durch Kommunikation, speziell Multimediakommunikation nicht bzw. nur ansatzweise berücksichtigt.
- Auch die Eigenentwicklung von Diensten für das Energiemanagement durch den Nutzer selbst wird in Projekten [11] behandelt, allerdings beschränken sie sich auf reine Konfiguration und lassen ebenfalls Kommunikationsdienste außen vor.
- Die Durchführung notwendiger Tests vor Einsatz der vom Nutzer konfigurierten Dienste wird komplett außer Acht gelassen.
- Zwar gibt es Lösungen für die automatisierte energetische Optimierung eines einzelnen Haushalts bzw. eines Clusters von Haushalten, wobei diese allerdings normalerweise von einem zentralen oder kombiniert zentralen/dezentralen Ansatz für die intelligente Steuerung ausgehen [11; 4; 9; 12]. Eine komplett dezentrale Lösung ist zwar komplexer, würde aber unbestreitbare Vorteile bei der Skalierbarkeit, der Ausfallsicherheit sowie insbesondere der individuellen, auch auf Privatinitiative basierenden Vernetzbarkeit

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0241

bieten. Ein entsprechender Ansatz mit allerdings nur eingeschränktem Anwendungsbereich ist bisher nur in [13] zu finden.

- Zudem ist bei bekannten Lösungen [11] der Verteilnetzbetreiber mit seiner Leitstelle direkt eingebunden. Nachteil hierbei, insbesondere im Hinblick auf die Sicherheit, ist, dass damit die Kommunikationsinfrastruktur der Kunden und des Verteilnetzbetreibers nicht mehr getrennt sind und damit Angriffe auf das Energienetz nicht von vornherein verhindert werden können. Eine Cluster-Lösung zur energetischen Optimierung ohne Kommunikationsschnittstelle zum Elektroenergiesystem wäre deutlich vorteilhafter.
- Ebenfalls bisher nicht betrachtet wurde die eigenständige Cluster-Bildung von Haushalten, d.h. wie sich auf komfortable Art und Weise Haushalte zu Clustern für das Energiemanagement zusammenschließen. Bisherige Lösungen [11; 9; 12] gehen von einer Cluster-Organisation durch eine zentrale Instanz aus. Vorteilhaft wäre hier eine Vernetzung auf Basis einer freiwilligen Energie-Community, um die Selbstorganisation der Vernetzung zu ermöglichen, Fragen des Datenschutzes zu klären, gemeinsame Interessen der Haushalte zu definieren etc.

Um die genannten Lücken zu schließen, wurde im Oktober 2011 das BMBF-FHprofUnt 2011-Projekt „Easy - Service Creation for Home and Energy Management (e-SCHEMA)“ gestartet [14; 15; 16]. Hierbei werden vier aufeinander aufbauende, im Folgenden beschriebene Lösungsansätze zum Thema „Haushalte im Smart Grid“ verknüpft, wobei alle dem Bereich „Smart Market“ zuzuordnen sind.

2 Diensteplattform für Haushalte

Im ersten Teil dieser neuen Gesamtlösung kann der Nutzer über ein z.B. im DSL-Router integriertes Service Management Framework (SMF) [17; 18] selbst entsprechend seinen persönlichen Anforderungen mittels SCE (Service Creation Environment) Mehrwertdienste für das Energiemanagement und weitere Smart Home-Anwendungen designen, z.B. durch grafische Verknüpfung von Apps der Haushaltsgerätehersteller (**Bild 1**). Im einfachsten Fall konfiguriert er zeitliche Abläufe, z.B. wann die Waschmaschine starten soll. Dies kann aber auch abhängig sein von einem entsprechenden Preissignal bei wechselnden Stromtarifen. Der personalisierte Energiemanagementdienst wird auf einer SDP (Service Delivery Platform) ausgeführt. Die für die Dienste benötigten Messwerte werden vom angebundenen Smart Meter bereitgestellt. Auf dieser Basis liefert das aus SCE und SDP bestehende SMF auch Informationen über den erfolgten und prognostizierten Energieverbrauch. Der Vorteil dieser Teillösung ist, dass Komfortempfinden und Rechte des Nutzers jederzeit berücksichtigt werden, Nachteil ist, dass die Energieverbrauchsoptimierung nur aus Sicht des einzelnen Nutzers stattfindet.

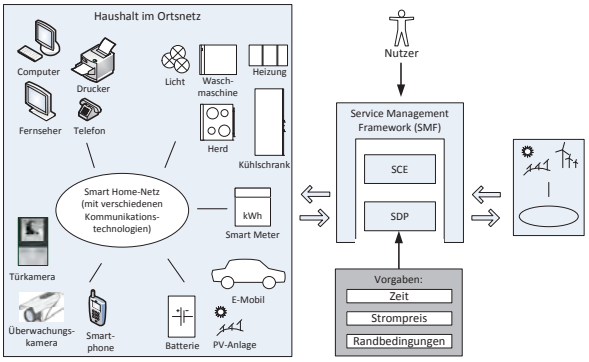


Bild 1 Service Management Framework zur Steuerung dezentraler Energieverbraucher und -erzeuger

In einem zweiten Schritt stellt daher das SMF in Bild 1 einen speziellen Dienst zur Verfügung, der selbständig unter Berücksichtigung von Nutzervorgaben (z.B. keine zeitlichen Einschränkungen für das Kochen, Minimierung der Stromkosten, maximal zulässige Leistungsaufnahme) aus z.B. Apps einen Mehrwertdienst komponiert, der den Energieverbrauch entsprechend den vorgegebenen Randbedingungen optimiert. Der resultierende Dienst sorgt dann beispielsweise dafür, dass die Lastkurve des Haushalts unter den gegebenen Randbedingungen so konstant wie möglich verläuft, um teure Leistungsspitzen zu vermeiden. Hier erfolgt die Energieverbrauchsoptimierung nicht nur aus Sicht des Nutzers, sondern des Haushalts, wobei sie besonders wirkungsvoll ist, wenn nicht nur Energieverbraucher, sondern auch z.B. zur Kompensation von Verbrauchsplateaus -erzeuger und/oder -speicher im Haushalt vorhanden sind.

In beiden Fällen geht die Leistungsfähigkeit des SMF deutlich über die derzeitiger Smart Home-Steuerungen hinaus, indem sie nicht nur das beschriebene Energiemanagement und IT-Dienste (z.B. Kalenderzugriff vor dem Laden des Elektrofahrzeugs) integriert, sondern auch Kommunikationsanwendungen (z.B. SMS oder Anruf, wenn die Waschmaschine fertig gewaschen hat). Da das SMF auf jeden Fall via IP an externe Netze wie Internet, Fest- und Mobilfunknetze angebunden ist, bieten sich als Applikationsprotokolle SIP (Session Initiation Protocol) für die Signalisierung und z.B. RTP (Real-time Transport Protocol) für den Nutzdatenaustausch an, d.h. die typischen Protokolle für die Multimedia over IP-Kommunikation [16]. Insofern handelt es sich beim SMF um die Kombination einer Smart Home-Steuerung und eines SIP Application Servers, wobei die Realisierung auf Basis OSGi (Open Services Gateway initiative) [23] erfolgt.

3 P2P-vernetzte Haushalte und Energie-Community

Die unabhängige Optimierung einzelner Haushalte führt nicht automatisch zu einer Optimierung für das Orts- bzw. Verteilnetz. Weitaus wirkungsvoller ist das Zusammen-

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0242

spiel vieler Haushalte. Daher werden bei der dritten Lösung die SMFs der einzelnen Haushalte vernetzt, vorteilhafterweise ohne zentrale Instanz mittels SIP-basiertem Peer-to-Peer-Netzwerk (P2P) [19] (**Bild 2**). Die Peers repräsentieren Haushalte, aber auch Energielieferanten, Verteilnetzbetreiber und andere Marktteilnehmer. Die Kommunikation ist anonym und verschlüsselt, zudem gibt jeder Peer nur das an Informationen weiter, was der Nutzer möchte. Zwischen den Peers auszutauschende Informationen sind z.B. der aktuelle Verbrauch, die Prognose, Kennzahlen zu Energieerzeugern und -speichern (auch von Elektrofahrzeugen) sowie Strompreise und Netzentgelte. Auf dieser Basis bietet die SMF einen speziellen Dienst bzw. Algorithmus, der selbständig unter Berücksichtigung der Informationen von allen anderen Peers den Energieverbrauch gemeinsam für das ganze P2P-Netzwerk (bis hin zum kompletten Ortsnetz) optimiert. Die hierfür notwendige Datenhaltung erfolgt verteilt auf Basis von DHT (Distributed Hash Table) und dem Chord-Algorithmus [24].

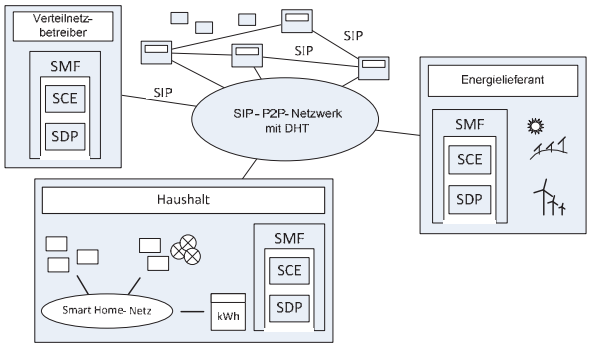


Bild 2 Peer-to-Peer-Netzwerk für das Haushaltsübergreifende Energiemangement

Bisher nicht gelöst ist, wie die Nutzer komfortabel dem Netzwerk beitreten und sich auf eine gemeinsame rechtliche Basis einigen. Dies kann vorteilhaft und neu in Form einer Energie-Community geschehen, d.h. die Vernetzung der Haushalte erfolgt durch Beitritt der Nutzer zu einem sozialen Netzwerk für Energie-Peers (**Bild 3**). Daher bietet die SMF auch Funktionalität für die Energie-Community. Die Mitglieder haben gleiche Interessen und ökonomische Ziele (z.B. Energieverbrauchsreduzierung, Kostenersparnis, Bewahrung der Umwelt), ihre Profile könnten den Energieverbrauch sowie vorhandene Speicher- und Erzeugungskapazität enthalten. Die Bildung von Subcommunities ist möglich (z.B. für Straße, Ortsteil, Ortsnetzstation, Kunden eines Energielieferanten oder Verteilnetzbetreibers). Die gesamte Community oder auch eine Subcommunity kann geschlossen auftreten und z.B. Regelenergie anbieten. Zudem sind Wettbewerbe zwischen Subcommunities z.B. mit dem Ziel einer Energieverbrauchsreduzierung denkbar, bei denen u.a. vergleichbare Haushalte ihre Verbrauchswerte austauschen. Die Energieverbrauchsoptimierung erfolgt aus Sicht der gesamten Community, auch Ortsnetz-übergreifend. Insofern repräsentieren die beiden kombinierten Ansätze des P2P-Netzwerkes und der Energie-Community eine umfassende neue Lösung zur Schaffung eines Smart Markets für Haushalte.

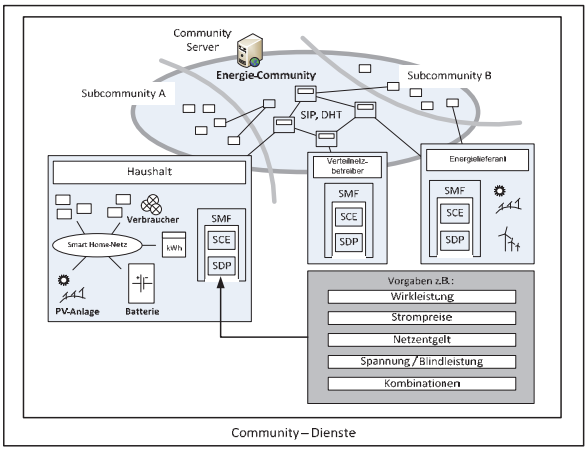


Bild 3 Energie-Community

Betreiber des Community Servers könnte z.B. eine Stadt oder ein Landkreis sein, die oder der ambitioniert das übergeordnete Ziel Klimaschutz verfolgt (vgl. das Förderprogramm „Masterplan 100% Klimaschutz“ des Bundesumweltministeriums). In Form der Energie-Community mit P2P-Vernetzung würde hierfür ein sehr leistungsfähiges Instrumentarium zur Verfügung stehen, das zusätzlich Preis- und Kostenreduktionspotenziale bietet.

4 Anwendungsszenarien

Die gemäß Kapitel 3 in einer Energie-Community bzw. in Subcommunities vernetzten Haushalte können entsprechend der in **Tabelle 1** genannten verschiedenen Anwendungsszenarien vorteilhaft zusammenwirken. Dabei nutzen sie die Flexibilisierungsmöglichkeiten, die sich durch den Einsatz dezentraler Speicher, Erzeuger und/oder zeitversetzt einzuschaltender Verbraucher (Demand Side Integration (DSI)) auftun. Je nach Anwendungsfall können sich dadurch Vorteile für den Verteilnetzbetreiber, den Bilanzkreisverantwortlichen, Energielieferanten, die Verbraucher oder auch die Umwelt etc. ergeben. Um die Wirksamkeit der skizzierten Energie-Community aufzuzeigen, werden im Folgenden einige konkrete, simulierte Anwendungsszenarien diskutiert.

Den Simulationen liegen folgende Parameter zugrunde: 1- bis 5-Personenhaushalte, anhand von Datenblättern bzw. Messungen ermittelte bzw. geschätzte Werte für Verbraucher (Kochen, Spülen, Kühlen, Gefrieren, Beleuchtung, Bürotechnik, Unterhaltungselektronik, Waschen, Trocknen, Warmwasserbereitung sowie Verschiedenes wie z.B. Bügeln), Erzeuger wie Photovoltaik-Anlagen (PV) und Blockheizkraftwerke (BHKW) und Batteriespeicher sowie Wahrscheinlichkeitsverteilungen zu Haushalten, Geräteverfügbarkeiten und deren Nutzung [20; 21].

Beitrag der Energie-Community	Im Haushalt angewandte Betriebsmittel	Vorteil	Hauptnutznießer	Community-Typ, Subcommunity
Vermeidung von Lastspitzen	Speicher und/oder DSI und/oder Erzeuger	Kostenersparnis bei Netzentgelten, weniger Netzausbau erforderlich	Verteilnetzbetreiber	regionale Subcommunity für Haushalte eines Verteilnetzbetreibers
Vermeidung von Regelenergie	Speicher und/oder DSI und/oder Erzeuger	Kostenersparnis bei Regelenergie	Bilanzkreisverantwortlicher	regionale Subcommunity für Haushalte eines Bilanzkreises
Vermeidung von „Überlastung der Netzbetriebsmittel“	Speicher und/oder DSI und/oder Erzeuger	weniger Netzausbau erforderlich	Verteilnetzbetreiber	lokale Subcommunity für Haushalte an einer Ortsnetzstation des Verteilnetzbetreibers
Optimierter Strompreis	Speicher und/oder DSI und/oder Erzeuger und/oder Informationen aus Community	möglichst günstiger Stromeinkauf	Energielieferant + Verbraucher	überregionale Subcommunity für Haushalte eines Energielieferanten
Spannungshaltung	Speicher und/oder DSI und/oder Erzeuger	weniger Netzausbau erforderlich	Verteilnetzbetreiber	lokale Subcommunity für Haushalte an einer Ortsnetzstation des Verteilnetzbetreibers
Anreize zum Energiesparen	Informationen aus Community	geringere Energiekosten	Verbraucher	überregionale Subcommunity für ähnlich strukturierte Haushalte oder Haushalte mit ähnlichen Interessen
Anreize zur CO_2-Reduzierung	Informationen aus Community	weniger Energie aus fossilen Brennstoffen	Umwelt	überregionale Subcommunity für Haushalte mit Ziel CO_2-Reduzierung
Abgestimmtes Laden von Elektrofahrzeugen	Informationen aus Community	weniger Netzausbau erforderlich, geringere Energiekosten	Verteilnetzbetreiber + Verbraucher	regionale Subcommunity für Haushalte mit Elektrofahrzeugen
u.a.				

Tabelle 1 Anwendungsszenarien für Energie-Community

Als Batteriespeicher kommt ein handelsübliches Modell mit 5400Wh max. und 1400Wh min. Ladekapazität, 4800W max. Lade- und 4000W max. Entladeleistung sowie 97% Wirkungsgrad beim Laden und Entladen zum Einsatz. Die eingesetzte PV-Anlage liefert 4000W Peak-Leistung. Für DSI wurden ausschließlich Waschmaschinen, Wäschetrockner und Geschirrspülmaschinen herangezogen. Darüber hinaus ist noch darauf hinzuweisen, dass die Simulationen nur 24 Stunden betrachten, d.h. Geräte werden erst ab 0 Uhr gestartet bzw. laufen nicht länger als 24 Uhr, zudem sind die Speicher um 0 Uhr zu 30% geladen.

Um die Leistungsfähigkeit der Simulation aufzuzeigen, wurde in einem ersten Schritt der reine Leistungsverbrauch für 1000 Haushalte bzw. 2024 Personen bei einer Verteilung von 39,7% 1-, 34,6% 2-, 12,7% 3-, 9,6% 4- und 3,4% 5-Personen-Haushalte [20] ermittelt und mit dem Standardlastprofil H0 für einen Werktag im Sommer gemäß [22] verglichen. Das Ergebnis geht aus **Bild 4** hervor. Die gute Übereinstimmung zwischen beiden Kurven zeigt die Qualität des zugrundegelegten Simulationsmodells, zumal die Berechnung für unendlich viele Haushalte eine noch bessere Übereinstimmung ergibt.

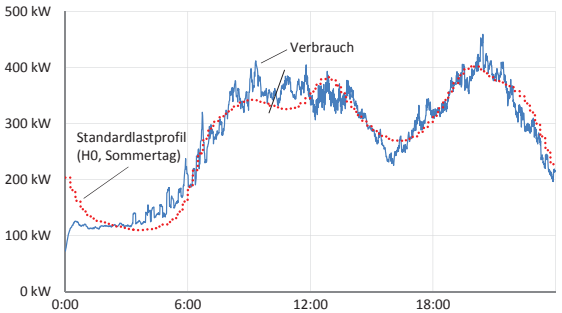


Bild 4 Für 1000 Haushalte simulierte Lastkurve sowie H0-Standardlastprofil für einen Werktag im Sommer

Szenario 1 behandelt die „Vermeidung von Lastspitzen“ mittels Batteriespeichern für eine Community von 1000 Haushalten. Das in **Bild 5** dargestellte Ergebnis zeigt, dass für eine 20%-ige Spitzenlastreduktion (459284W → 367427W) bereits 23 Batteriespeicher ausreichen, d.h. nur 2,3% der Haushalte mit jeweils einem Batteriespeicher ausgerüstet werden müssten. Erwähnenswert hierbei ist,

dass nicht die Kapazität der Batteriespeicher, sondern die maximale Entladeleistung der begrenzende Faktor ist.

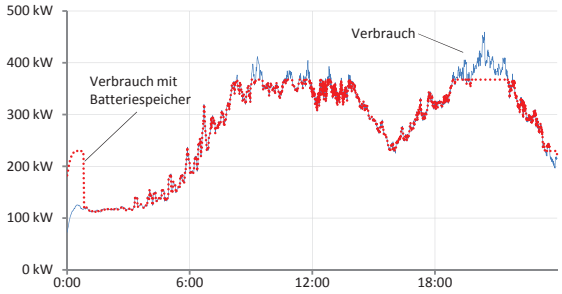


Bild 5 Simulation der Lastkurve für 1000 Haushalte und 23 Batteriespeicher mit 20% Spitzenlastabsenkung

Im Vergleich wird in Szenario 2 das gleiche Thema „Vermeidung von Lastspitzen“ mit DSI angegangen. Wiederum wurde der Verbrauch von 1000 Haushalten zugrundegelegt. Gemäß **Bild 6** ist bei Einbeziehung von 50% aller Wasch-, Spülmaschinen und Wäschetrockner eine Spitzenlastreduktion von 10% (459284W → 413356W) allein durch DSI erreichbar.

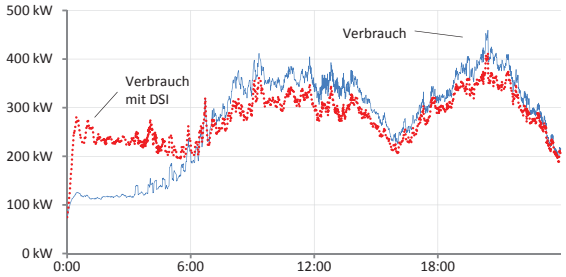


Bild 6 Simulation der Lastkurve für 1000 Haushalte und 50% DSI mit 10% Spitzenlastabsenkung

Szenario 3 für die „Vermeidung von Lastspitzen“ kombiniert nun Batteriespeicher und DSI. Das Ergebnis für 1000 Haushalte und 20% Spitzenlastreduktion (459284W → 367427W) zeigt **Bild 7**. In diesem Anwendungsfall sind 18 Batteriespeicher (1,8%) und 20% DSI notwendig.

Insgesamt zeigt sich in den Szenarien 1 bis 3, dass Batteriespeicher für die Spitzenlastreduzierung besonders wirksam sind.

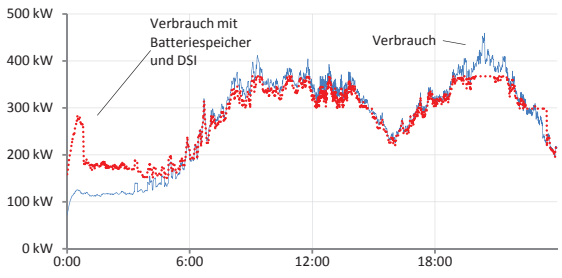


Bild 7 Simulation der Lastkurve für 1000 Haushalte, 18 Batteriespeicher und 20% DSI mit 20% Spitzenlastabsenkung

Ein zweiter Schwerpunkt in Tabelle 1 ist die „Vermeidung von Überlastung der Netzbetriebsmittel“. Dies wird in den folgenden Szenarien 4 und 5 konkret am Beispiel der Leistungsbelastung eines Ortsnetztransformators behandelt. Dabei werden bezüglich des Verbrauchs 50 Haushalte mit 102 Personen (20 1-, 17 2-, 6 3-, 5 4- und 2 5-Personen-Haushalte) pro Ortsnetzstation zugrunde gelegt. Zudem speisen 22 PV-Anlagen mit jeweils 4kWp ein. D.h., es wird von einer Ortsnetzstation auf dem Land mit einer sehr hohen PV-Konzentration ausgegangen, 44% der Haushalte haben eine PV-Anlage. Dies führt dazu, dass um die Mittagszeit ca. das Doppelte (-71649W) der Spitzenlast (35663W) aus den Abendstunden ins Netz eingespeist wird. Geht man davon aus, dass der Ortsnetztrafo so dimensioniert wurde, dass normalerweise 70% Leistung nicht überschritten werden, wäre er durch die PV-Einspeisung zu 140% be- und damit überlastet.

Dem wird in Szenario 4 nach **Bild 8** durch den Einsatz von 10 Batteriespeichern entgegengewirkt. Dies bedeutet, dass durch diese Maßnahme die 40%-ige Überlastung auf 0% zurückgefahren werden kann. Dazu müssen 20% der Haushalte bzw. näherungsweise die Hälfte der PV-Anlagen mit je einem Batteriespeicher ausgestattet werden.

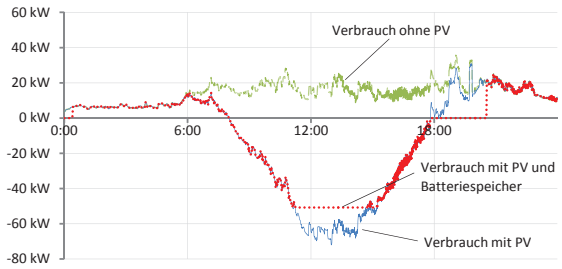


Bild 8 Simulation der Leistungskurve für 50 Haushalte, 22 PV-Anlagen und 10 Batteriespeicher an einer Ortsnetzstation

Szenario 5 gemäß **Bild 9** liefert das gleiche Endergebnis durch Kombination von 8 Batteriespeichern und 24% DSI.

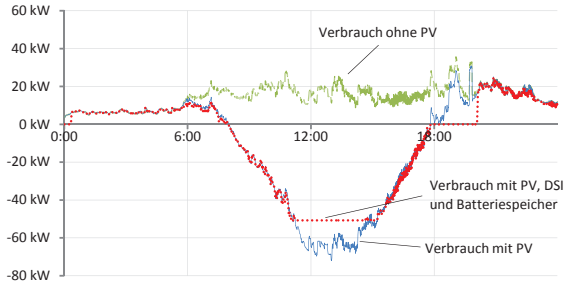


Bild 9 Simulation der Leistungskurve für 50 Haushalte, 22 PV-Anlagen, 8 Batteriespeicher und 24% DSI an einer Ortsnetzstation

Auch in den Szenarien 4 und 5 zeigt sich wiederum die besondere Wirksamkeit des Einsatzes von Batteriespei-

chern mit dem Nachteil der zusätzlichen Kosten, wobei diese aber zumindest zum Teil vorteilhafterweise auch von Elektrofahrzeugen bereitgestellt werden können.

Weitere in Tabelle 1 genannte Anwendungsfälle greifen die Themen Energiesparen und CO_2-Reduzierung auf. Hier könnten z.B. über die Community gelieferte Daten zum Energieverbrauch vergleichbarer Haushalte, zu CO_2-Äquivalenten der aktuell bezogenen Energie, zu Möglichkeiten der energetischen Modernisierung, Effizienz und Optimierung sowie zu Fördermöglichkeiten und Prämien (vgl. z.B. Aktivitäten der Stadt Frankfurt) in der Community gestreut und in der Folge wirksam werden. Darüber hinaus könnten spezielle Subcommunities von Nutzern, Herstellern und Serviceanbietern von Elektrofahrzeugen sowie dem Verteilnetzbetreiber deren Einführung und Handhabung erleichtern.

5 Zusammenfassung und Ausblick

Mit dem Konzept der personalisierten Dienste und vor allem der P2P-vernetzten Energie-Community wird eine neue und sehr leistungsfähige Lösung für das Smart Home und vor allem den Smart Market im Bereich des Verteilnetzes vorgestellt und durch erste Simulationsergebnisse bestätigt. Vorteile zeichnen sich nicht nur für die Kunden, sondern für alle Teilnehmer an der Wertschöpfungskette ab. Weitere Untersuchungen, u.a. zu den verschiedenen Anwendungsszenarien und gegenseitigen Abhängigkeiten, aber auch zur Wirtschaftlichkeit und vor allem zu Sicherheit und Datenschutz sind noch notwendig.

Das dieser Publikation zugrunde liegende Vorhaben wird mit Mitteln des Bundesministeriums für Bildung und Forschung (BMBF) unter dem Förderkennzeichen 17018A/B11 gefördert. Die Verantwortung für den Inhalt dieser Veröffentlichung liegt bei den Autoren.

6 Literatur

[1] Köhler-Schute, C.: Smart Grids. Aus „Smart Grids – Die Energieinfrastruktur im Umbruch“. KS-Energy-Verlag, Januar 2012, S. 16-43

[2] Bundesnetzagentur; “Smart Grid” und “Smart Market” – Eckpunktepapier der Bundesnetzagentur zu den Aspekten des sich verändernden Energieversorgungssystems. Bonn, Dezember 2011

[3] Grönner, J.: „Smart Country“. 4. Göttinger Energietagung “Dezentralisierung und Netzausbau”, 22.-23.03.2012

[4] www.regmodharz.de

[5] Tröschel, M.: Aktive Einsatzplanung in holonischen Virtuellen Kraftwerken. Dissertation Universität Oldenburg, 02.11.2010

[6] Stiegler, M.; Klöker, P.; Wrede, S.: Zukünftige Herausforderungen und Lösungsansätze für Mittel- und Niederspannungsnetze – Erfolgsfaktor Automatisierung. Aus „Smart Grids – Die Energieinfrastruktur im Umbruch“. KS-Energy-Verlag, Januar 2012, S. 44-54

[7] Graßmann, N.; Kreibich, C.: Energierechtliche Rahmenbedingungen für Smart Grids. Aus „Smart Grids – Die Energieinfrastruktur im Umbruch“. KS-Energy-Verlag, Januar 2012, S. 116-126

[8] Gutzwiller, R.; Koch, S.; Meier, D.; Wiederkehr, M.: Lokales Lastmanagement – Stromverbrauch sinnvoll mit der Erzeugung koordinieren. Bulletin SEV/AES 22/23 2008, S. 9-13

[9] Weidlich, Anke et al.: Smart Houses Interacting with Smart Grids to achieve next-generation efficiency and sustainability. Deliverable D5.5 EU project Smart House/Smart Grid, 31.10.2011

[10] www.connected-living.org/ziele/sercho_technologie/

[11] Hauser, Eva et al.: Modellstadt Mannheim – Arbeitspaket 5: Untersuchung des technischen, energiewirtschaftlichen und regulatorischen Rahmens. BMWi-Projekt im Rahmen E-Energy, 05.07.2011

[12] www.smarthouse-smartgrid.eu/

[13] Kamper, A.: Dezentrales Lastmanagement zum Ausgleich kurzfristiger Abweichungen im Stromnetz. Dissertation, Karlsruher Institut für Technologie, 27.07.2009

[14] www.ecs.hs-osnabrueck.de/e-schema.html

[15] Steinheimer, M.; Trick, U.; Ruhrig, P.: Energy communities in Smart Markets for optimisation of peer-to-peer interconnected Smart Homes. Proc. CSNDSP 2012, Poznan, 18.-20.07.2012

[16] Trick, U.; Steinheimer, M.; Ruhrig, P.; Tönjes, R.; Hölker, D.; Fischer, M.: Herausforderungen an die Kommunikationstechnik im Smart Home/Grid. VDE/ITG Fachtagung Mobilkommunikation 2012, Osnabrück, Mai 2012

[17] Lehmann, A.; Eichelmann, T.; Trick, U.; Lasch, R.; Ricks, B.; Tönjes, R.: TeamCom: A Service Creation Platform for Next Generation Networks. Proc. of ICIW 2009, Venice, pp. 12-17

[18] Eichelmann, T.; Fuhrmann, W.; Trick, U.; Ghita, B.: Discussion on a framework and its service structures for generating JSLEE based value-added services. Proc. of ITA 2011, Wrexham, pp. 169-176

[19] Lehmann, A:; Fuhrmann, W.; Trick, U.; Ghita, B.: New possibilities for the provision of value-added services in SIP-based peer-to-peer networks. Proc. of SEIN 2008, Wrexham, pp. 167-176

[20] Statistisches Bundesamt: Bauen und Wohnen – Mikrozensus – Zusatzerhebung 2010. Fachserie 5, Heft 1, April 2012

[21] Hölker, D.: e-SCHEMA - Simulation. Int. Bericht, Hochschule Osnabrück, 23.08.2012

[22] E.ON Mitte: Normierte Standardlastprofile H0, L0-L2, G0-G6 für Hessen. E.ON Mitte AG, 2011

[23] Alves, A.: OSGi in Depth. Manning, 2011

[24] Stoica, I. et al.: Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. IEEE/ACM Transactions on Networking, Vol. 11, Issue 1, 2003, pp. 17-32

Appendix Alves 06

HAL EN Sign in
HAL Portal theses + Upload
HAL theses
thèses en ligne
Homepage Browse HAL documentation

Download

Theses Year : 2013

Embedding security policies into pervasive computing systems

en fr

Pengfei Liu (1)

Show details

1 LaBRI - Laboratoire Bordelais de Recherche en Informatique

Abstract en fr

When developing pervasive computing applications, it is critical to specify security policies and develop security mechanisms to ensure the confidentiality and integrity of the applications. Numerous policy specification languages only focus on their expressive power. The emerging challenges in pervasive computing systems can not be fulfilled by these approaches. For instance, context awareness is a central aspect of pervasive computing systems. Existing approaches rarely consider context information in their language.This thesis proposes a generative approach dedicated to specifying and enforcing security policies in pervasive computingapplications. To specify a policy, we propose a context-aware policy specification language which helps developers to specify policy rules and required entities (e.g. spatial description, roles, context information). Policies are implemented by term rewriting systems which offers great verification power. To enforce a policy, we propose an architecture that embeds important concepts of security policies (subject, object, security related context) into pervasive computing applications. To apply our approach, we enriched an existing approach which is dedicated to develop pervasive computing applications. Based on the policy specification and the enriched pervasive computing application descriptions, a dedicated programming framework is generated. This framework guides the implementation and raises the level of abstraction which can reduce the workloads of developers.

Keywords en fr
Security policy
Domain-Specific Language
Pervasive computing system
Generative programming

Domains
Other [cs.OH]

Complete list of metadata

Dates and versions

tel-00799284 , **version 1** (12-03-2013)

Identifiers

HAL Id : **tel-00799284 , version 1**

Cite

Pengfei Liu. Embedding security policies into pervasive computing systems. Other [cs.OH]. Université Sciences et Technologies - Bordeaux I, 2013. English. (NNT : 2013BOR14735). (tel-00799284)

Export

BibTeX XML-TEI Dublin Core DC Terms EndNote DataCite

Collections

CNRS STAR

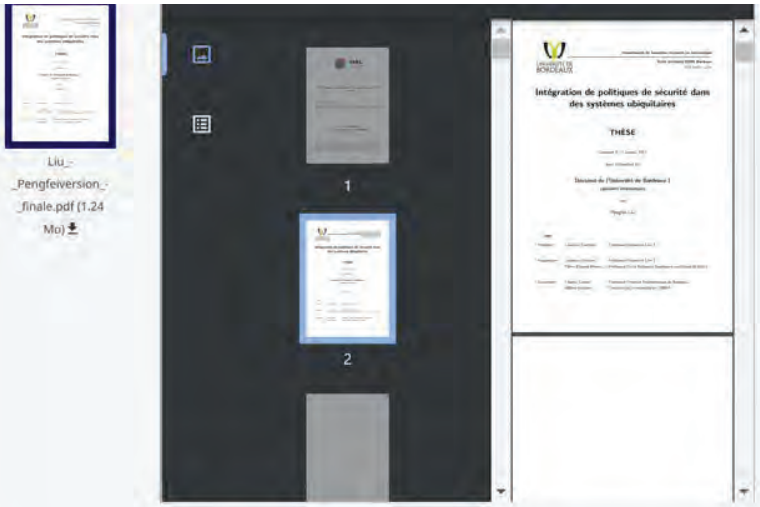
331 View 457 Download

Share

Files and preview

Fichier principal





Cited literature [97 references]

Filter

International Workshop on Policies for Distributed Systems and Networks, POLICY '01, pp.165-182, 2001.

Chercher sur Google Scholar

A. De-castro-alves, **OSGi in Depth**, 2011.

Chercher sur Google Scholar

A. S. De-oliveira, **Réécriture et Modularité pour les Politiques de Sécurité**, 2008.

Chercher sur Google Scholar

A. K. Dey, G. D. Abowd, and D. Salber, **A Context-Based Infrastructure for Smart Environments**, 1999.

10.1007/978-1-4471-0743-9_11 Chercher sur Google Scholar

W. K. Edwards and R. E. Grinter, **At Home with Ubiquitous Computing: Seven Challenges**, *Proceedings of the 3rd international conference on Ubiquitous Computing, UbiComp '01*, pp.256-272, 2001.

ABES STAR Contact

Contact
Support

Resources
Documentation
Training
FAQ
API
OAI-PMH
AuréHAL

Informations
About
Privacy and personal data
Legal terms
Accessibility
Declaration of compliance

Legal issues
I publish, what are my rights?
The Digital Republic bill
Non-assignment strategy of rights
SHERPA/RoMEO

Portals
Institutional portals
HAL
HAL SHS
HAL Thèses
MédiHAL

CCSD
CCSD
Episciences
Sciencesconf

Manage services 0



Embedding security policies into pervasive computing systems

Pengfei Liu

▶ **To cite this version:**

Pengfei Liu. Embedding security policies into pervasive computing systems. Other [cs.OH]. Université Sciences et Technologies - Bordeaux I, 2013. English. NNT : 2013BOR14735 . tel-00799284

HAL Id: tel-00799284

https://theses.hal.science/tel-00799284v1

Submitted on 12 Mar 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0250

UNIVERSITÉ DE BORDEAUX

Département de formation doctorale en informatique

École doctorale EDMI Bordeaux
N° d'ordre : 4735

Intégration de politiques de sécurité dans des systèmes ubiquitaires

THÈSE

soutenue le 17 Janvier 2013

pour l'obtention du

Doctorat de l'Université de Bordeaux 1

(spécialité informatique)

par

Pengfei Liu

Jury

Président :	Laurence Duchien,	Professeur Université Lille 1
Rapporteurs :	Laurence Duchien,	Professeur Université Lille 1
	Pierre-Etienne Moreau,	Professeur Ecole Nationale Supérieure des Mines de Nancy
Encadrants :	Charles Consel,	Professeur l'Institut Polytechnique de Bordeaux
	Hélène Kirchner,	Directrice de la recherche de l'INRIA

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0251

Cette thése est librement téléchargeable sur hal.inria.fr

Copyright © 2013, Pengfei Liu,

LaBRI
UnitMixte de Recherche CNRS (UMR 5800),
351 cours de la libération,
F-33405 Talence cedex

INRIA Bordeaux Sud-Ouest
Bâtiment A 29,
351 cours de la libération,
F-33405 Talence Cedex

Université de Bordeaux 1



Cette œuvre est mise disposition selon le contrat Attribution-ShareAlike 3.0 Unported disponible en ligne :

http://creativecommons.org/licenses/by-sa/3.0/

Premiére édition, janvier 2013.

BIBLIOGRAPHY

[1] Standard and extended x10 code protocol, 1993. URL http://software.x10.com/pub/manuals/xtdcode.pdf.

[2] E. Aarts, R. Harwig, and M. Schuurmans. *The invisible future: The seamless integration of technology into everyday life*. McGraw-Hill Companies, 2002.

[3] E. H. L. Aarts and S. Marzano. *The New Everyday View on Ambient Intelligence*. Uitgeverij 010 Publishers, 2003.

[4] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles. Towards a better understanding of context and context-awareness. In *Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*, HUC '99, pages 304–307, London, UK, 1999. Springer-Verlag.

[5] M. D. Abrams. Renewed understanding of access control policies. In *Proceeding of the 16th National Computer Security Conference*, pages 87–96, Baltimore, Maryland, USA, September 1993.

[6] D. Agrawal, J. Giles, K. won Lee, and J. Lobo. Policy ratification. In *Proceedings of the Sixth IEEE International Workshop on Policies for Distributed Systems and Networks*, pages 223–232. IEEE Computer Society, 2005.

[7] T. Ahmed and A. R. Tripathi. Static verification of security requirements in role based cscw systems. In *Proceedings of the eighth ACM symposium on Access control models and technologies*, SACMAT '03, pages 196–203, New York, NY, USA, 2003. ACM.

[8] F. Aldrich. Smart homes: Past, present and future. In *Inside the Smart Home*, pages 17–39. Springer London, 2003.

[9] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transaction on Dependable and Secure Computing*, 1(1):11–33, Jan. 2004.

[10] F. Baader and T. Nipkow. *Term rewriting and all that*. Cambridge University Press, New York, NY, USA, 1998.

[11] M. Backes, W. Bagga, G. Karjoth, and M. Schunter. Efficient comparison of enterprise privacy policies. In *Proceedings*

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0254

of the 2004 ACM symposium on Applied computing, pages 375–382. ACM Press, 2004.

[12] D. Balfanz, G. Durfee, D. Smetters, and R. Grinter. In search of usable security: five lessons from the field. *Security Privacy, IEEE*, 2(5):19 –24, sept.-oct. 2004.

[13] E. Balland, P. Brauner, R. Kopetz, P.-E. Moreau, and A. Reilles. Tom: Piggybacking rewriting on java. In *Conference on Rewriting Techniques and Applications*, volume 4533 of *LNCS*, pages 36–47, Paris, France, June 2007. Springer-Verlag.

[14] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice (2nd Edition)*. Addison-Wesley Professional, 2 edition, Apr. 2003.

[15] D. Benslimane, S. Dustdar, and A. Sheth. Services mashups: The new generation of web applications. *IEEE Internet Computing*, 12(5):13–15, Sept. 2008.

[16] M. Bishop. What is computer security? In *Security & Privacy, IEEE*, volume 1, pages 67–69, Davis, CA, USA, Jan 2003.

[17] M. Bishop. *Introduction to Computer Security*. Addison-Wesley Professional, 2004.

[18] A. Bouhoula and F. Jacquemard. Automatic verification of sufficient completeness for conditional constrained term rewriting systems. Rapport de recherche RR-5863, INRIA, 2006.

[19] T. Bourdier, H. Cirstea, M. Jaume, and H. Kirchner. Formal specification and validation of security policies. In *Foundations & Practice of Security*, volume 6888 of *Lecture Notes in Computer Science*, pages 148–163, Paris, France, 2011. Springer, Heidelberg.

[20] T. Braun, M. Diaz, J. Enrquez-Gabeiras, and T. Staub. *End-to-End Quality of Service Over Heterogeneous Networks*. Springer Publishing Company, Incorporated, 1 edition, 2008.

[21] D. Brewer and M. Nash. The chinese wall security policy. In *Security and Privacy, 1989. Proceedings., 1989 IEEE Symposium on*, volume 2, pages 206 –214, may 1989.

[22] G. Brose, A. Vogel, and K. Duddy. *Java Programming with CORBA, Third Edition*. John Wiley & Sons, Inc., New York, NY, USA, 3rd edition, 2001.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0255

[23] J. Bruneau, W. Jouve, and C. Consel. DiaSim: A Parameterized Simulator for Pervasive Computing Applications. In *6th International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (Mobiquitous 09)*, Toronto, Canada, 2009. IEEE.

[24] D. Cassou. *Développement logiciel orienté paradigme de conception: la programmation dirigée par la spécification*. PhD thesis, Université de Bordeaux, 2011.

[25] D. Cassou, B. Bertran, N. Loriant, and C. Consel. A generative programming approach to developing pervasive computing systems. *In GPCE 09: Proceedings of the 8th international conference on Generative programming and component engineering*, pages 137–146, october 2009.

[26] T. Chaari, F. Laforest, and A. Celentano. Adaptation in Context-Aware Pervasive Information Systems: The SECAS Project. *Int. Journal on Pervasive Computing and Communications(IJPCC)*, 3(4):400–425, Dec. 2007.

[27] H. Chen, F. Perich, T. Finin, and A. Joshi. Soupa: Standard ontology for ubiquitous ans pervasive applications. In *International Conference on mobile and ubiquitous system: Networking and Services*, 2004.

[28] S. Chetan and R. Campbell. Towards fault tolerant pervasive computing. In *Pervasive 2004 Workshop on Sustainable Pervasive Computing*, pages 38–44, 2004.

[29] R. C. Christopher, C. K. Hess, M. Roman, and R. H. Campbell. Gaia: A development infrastructure for active spaces. In *Workshop on Application Models and Programming Tools for Ubiquitous Computing (held in conjunction with the UBICOMP 2001*, 2001.

[30] D. D. Clark and D. R. Wilson. A comparison of commercial and military computer security policies. *Security and Privacy, IEEE Symposium on*, 0:184, 1987.

[31] C. Consel. DiaSuite:A Paradigm-Oriented Software Development Approach (invited paper). In *20th ACM SIGPLAN workshop on Partial evaluation and program manipulation : PEPM 11*, pages 77–78, Austin, TX, United States, Jan. 2011. ACM.

[32] D. J. Cook, M. Youngblood, and S. K. Das. *A Multi-agent Approach to Controlling a Smart Environment*, volume 4008, pages 165–182. Springer-Verlag, 2006.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0256

[33] N. Damianou, N. Dulay, E. Lupu, and M. Sloman. The ponder policy specification language. In *Proceedings of the International Workshop on Policies for Distributed Systems and Networks*, POLICY '01, pages 18–38, London, UK, 2001. Springer-Verlag.

[34] A. De Castro Alves. *OSGi in Depth*. Manning Publications, 2011.

[35] A. S. de Oliveira. *Réécriture et Modularité pour les Politiques de Sécurité*. PhD thesis, Université Henri Poincaré, 2008.

[36] A. K. Dey, G. D. Abowd, and D. Salber. A context-based infrastructure for smart environments, 1999.

[37] W. K. Edwards and R. E. Grinter. At home with ubiquitous computing: Seven challenges. In *Proceedings of the 3rd international conference on Ubiquitous Computing*, UbiComp '01, pages 256–272, London, UK, 2001. Springer-Verlag.

[38] M. T. et al. Extensible access control markup language(xacml) version 2.0. Technical report, OASIS, 2005.

[39] D. Ferraiolo and R. Kuhn. Role-based access control. In *15th NIST-NCSC National Computer Security Conference*, pages 554–563, 1992.

[40] K. Fisler, S. Krishnamurthi, L. A. Meyerovich, and M. C. Tschantz. Verification and change-impact analysis of access-control policies. In *Proceedings of the 27th international conference on Software engineering*, ICSE '05, pages 196–205, New York, NY, USA, 2005. ACM.

[41] M. Fowler. *Domain-Specific Languages (Addison-Wesley Signature Series (Fowler))*. Addison-Wesley Professional, 1 edition, Oct. 2010.

[42] R. Frohardt, B.-Y. E. Chang, and S. Sankaranarayanan. Access nets: modeling access to physical spaces. In *Proceedings of the 12th international conference on Verification, model checking, and abstract interpretation*, VMCAI'11, pages 184–198, Berlin, Heidelberg, 2011. Springer-Verlag.

[43] D. Garlan, D. Siewiorek, A. Smailagic, and P. Steenkiste. Project aura: Toward distraction-free pervasive computing. *IEEE Pervasive Computing*, 1:22–31, 2002.

[44] T. Genet. Decidable approximations of sets of descendants and sets of normal forms. In *9th Conference on Rewriting Techniques and Applications*, volume 1379 of *Lecture Notes in Computer Science*, pages 151–165, Tsukuba, Japan, 1998. Springer-Verlag.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0257

[45] S. Hadim and N. Mohamed. Middleware: Middleware challenges and approaches for wireless sensor networks. *IEEE Distributed Systems Online*, 7(3):1–16, Mar. 2006.

[46] R. Haux. Individualization, globalization and health about sustainable information technologies and the aim of medical informatics. *International Journal of Medical Informatics*, 75(12):795 – 808, 2006.

[47] R. Hayton, J. Bacon, and K. Moody. Access control in an open distributed environment. *IEEE Symposium on Security and Privacy*, 0:3–15, 1998.

[48] K. Henricksen, J. Indulska, and A. Rakotonirainy. Infrastructure for pervasive computing: Challenges. In *GI Jahrestagung (1)*, pages 214–222, 2001.

[49] S. Jajodia, P. Samarati, M. L. Sapino, and V. S. Subrahmanian. Flexible support for multiple access control policies. *ACM Transaction on Database Systems*, 26(2):214–260, jun 2001.

[50] S. Jajodia, P. Samarati, and V. S. Subrahmanian. A logical language for expressing authorizations. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, SP '97, pages 31–42, Washington, DC, USA, 1997. IEEE Computer Society.

[51] B. Johanson, O. Fox, and T. Winograd. The interactive workspaces project: Experiences with ubiquitous computing rooms. *IEEE Pervasive Computing*, 1:67–74, 2002.

[52] L. Kagal, T. W. Finin, and A. Joshi. A policy language for a pervasive computing environment. In *IEEE 4th International Workshop on Policies for Distributed Systems and Networks*, pages 63–74, June 2003.

[53] L. Kagal, V. Korolev, S. Avancha, A. Joshi, T. Finin, and Y. Yesha. Centaurus: an infrastructure for service management in ubiquitous computing environments. *Wireless Network*, 8(6):619–635, november 2002.

[54] L. Kagal, V. Korolev, H. Chen, A. Joshi, and T. W. Finin. Centaurus: A framework for intelligent services in a mobile environment. In *Proceedings of the International Workshop on Smart Appliances and Wearable Computing*, pages 195–201, 2001.

[55] L. Kagal, F. Perich, A. Joshi, and T. Finin. A security architecture based on trust management for pervasive computing systems. In *Grace Hopper Celebration of Women in Computing*, October 2002.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0258

[56] L. Kagal, J. Undercoffer, F. Perich, A. Joshi, T. Finin, and Y. Yesha. Vigil: Providing trust for enhanced security in pervasive systems. Technical report, University of Maryland Baltimore County, 2001.

[57] A. D. Keromytis, S. Ioannidis, M. B. Greenwald, and J. M. Smith. Scalable security policy mechanisms, 2001.

[58] A. Kissinger and J. C. Hale. Lopol: A deductive database approach to policy analysis and rewriting. In *Second Annual SELinux Symposium*, 2006.

[59] D. Lin, P. Rao, E. Bertino, N. Li, and J. Lobo. Exam - a comprehensive environment for the analysis of access control policies. Technical report, Dept of Computer Science, Purdue University, 2007.

[60] D. Maier and D. Warren. *Computing with logic: logic programming with Prolog*. Benjamin/Cummings Pub. Co., 1988.

[61] J. McLean. A comment on the "Basic Security Theorem" of Bell and LaPadula. *Information Processing Letters*, 20(2):67–70, 1985.

[62] S. Mokhtar, D. Fournier, N. Georgantas, and V. Issarny. Context-aware service composition in pervasive computing environments. In N. Guelfi and A. Savidis, editors, *Rapid Integration of Software Engineering Techniques*, volume 3943 of *Lecture Notes in Computer Science*, pages 129–144. Springer Berlin / Heidelberg, 2006.

[63] D. J. Moore, I. A. Essa, and M. H. H. Iii. Exploiting human actions and object context for recognition tasks. *Computer Vision, IEEE International Conference on*, 1:80, 1999.

[64] P. Naldurg, S. Schwoon, S. Rajamani, and J. Lambert. Netra:: seeing through access control. In *Proceedings of the fourth ACM workshop on Formal methods in security*, FMSE '06, pages 55–66, New York, NY, USA, 2006. ACM.

[65] J. Nehmer, M. Becker, A. Karshmer, and R. Lamm. Living assistance systems: an ambient intelligence approach. In *Proceeding of the 28th international Conference on Software Engineering*, pages 43–50. ACM Press, 2006.

[66] R. d. F. B. Neto and M. d. G. C. Pimentel. Toward a domain-independent semantic model for context-aware computing. In *Proceedings of the Third Latin American Web Congress*, LA-WEB '05, pages 61–, Washington, DC, USA, 2005. IEEE Computer Society.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0259

[67] U. S. D. of Defense. Trusted computer system evaluation criteria, December 1985.

[68] R. Orfali and D. Harkey. *Client/server programming with Java and CORBA (2nd ed.)*. John Wiley & Sons, Inc., New York, NY, USA, 1998.

[69] X. Ou and S. Govindavajhala. Mulval: A logic-based network security analyzer. In *14th USENIX Security Symposium*, pages 113–128, 2005.

[70] T. Parr. *The Definitive ANTLR Reference: Building Domain-Specific Languages*. Pragmatic Programmers. Pragmatic Bookshelf, first edition, May 2007.

[71] E. Pitt and K. McNiff. *Java.rmi: The Remote Method Invocation Guide*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[72] A. Ranganathan, S. Chetan, J. Al-Muhtadi, R. H. Campbell, and M. D. Mickunas. Olympus: A high-level programming model for pervasive computing environments. In *Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications*, PERCOM '05, pages 7–16, Washington, DC, USA, 2005. IEEE Computer Society.

[73] I. Ray, M. Kumar, and L. Yu. Lrbac: A location-aware role-based access control model. In *Proceedings of the 2nd International Conference on Information Systems Security*, pages 147–161, 2006.

[74] W. Reisig and G. Rozenberg, editors. *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1 edition, 1998.

[75] C. Ribeiro, A. Zuquete, P. Ferreira, and P. Guedes. Spl: An access control language for security policies and complex constraints. In *NDSS*, 2001.

[76] M. Román, C. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell, and K. Nahrstedt. A middleware infrastructure for active spaces. *IEEE Pervasive Computing*, 1(4):74–83, Oct. 2002.

[77] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. Sip: Session initiation protocol, 2002.

[78] F. Sadri. Ambient intelligence: A survey. *ACM Computing Survey*, 43(4):36:1–36:66, Oct. 2011.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0260

[79] D. Salber, A. K. Dey, and G. D. Abowd. The context toolkit: aiding the development of context-enabled applications. In *Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit*, CHI '99, pages 434–441, New York, NY, USA, 1999. ACM.

[80] G. Sampemane, P. Naldurg, and R. H. Campbell. Access control for active spaces. In *Proceedings of the 18th Annual Computer Security Applications Conference*, ACSAC '02, pages 343–, Washington, DC, USA, 2002. IEEE Computer Society.

[81] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *Computer*, 29:38–47, February 1996.

[82] B. Sarna-Starosta and S. D. Stoller. Policy analysis for security-enhanced linux. In *Proceedings of the 2004 Workshop on Issues in the Theory of Security (WITS)*, pages 1–12, April 2004. Available at http://www.cs.sunysb.edu/~stoller/WITS2004.html.

[83] M. Satyanarayanan. Pervasive computing: Vision and challenges. *IEEE Personal Communications*, 8:10–17, 2001.

[84] P. Schneider-Kamp, J. Giesl, A. Serebrenik, and R. Thiemann. Automated termination analysis for logic programs by term rewriting. In *Proceedings of the 16th international conference on Logic-based program synthesis and transformation*, LOPSTR'06, pages 177–193, Berlin, Heidelberg, 2007. Springer-Verlag.

[85] E. Serral, P. Valderas, and V. Pelechano. Towards the model driven development of context-aware pervasive systems. *Pervasive and Mobile Computing*, 6(2):254–280, Apr. 2010.

[86] N. Shadbolt. Ambient intelligence. *IEEE Intelligent Systems*, 18(4):2–3, July 2003.

[87] A. Singh, C. R. Ramakrishnan, I. V. Ramakrishnan, S. D. Stoller, and D. S. Warren. Security policy analysis using deductive spreadsheets. In *Proceedings of the 2007 ACM workshop on Formal methods in security engineering*, FMSE '07, pages 42–50, New York, NY, USA, 2007. ACM.

[88] M. Sloman and E. Lupu. Engineering policy-based ubiquitous systems. *Computing Journal*, 53(7):1113–1127, Sept. 2010.

[89] J. a. P. Sousa and D. Garlan. Aura: an architectural framework for user mobility in ubiquitous computing environments. In *Proceedings of the IFIP 17th World Computer*

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0261

Congress, WICSA 3, pages 29–43, Deventer, The Netherlands, The Netherlands, 2002. Kluwer, B.V.

[90] R. N. Taylor, N. Medvidovic, and E. M. Dashofy. *Software Architecture: Foundations, Theory, and Practice*. Wiley, 2009.

[91] D. Thomas. Mda: Revenge of the modelers or uml utopia. *IEEE Software*, 21:15–17, 2004.

[92] K. Vaniea, Q. Ni, L. Cranor, and E. Bertino. Access control policy analysis and visualization tools for security professionals. In *USM 08: Workshop on Usable IT Security Management 2008*, Carnegie Mellon University, Pittsburgh, July 2008.

[93] D. C. Verma. *Policy-Based Networking: Architecture and Algorithms*. New Riders Publishing, Thousand Oaks, CA, USA, 2000.

[94] T. D. Wang, B. Parsia, and J. Hendler. A survey of the web ontology landscape. In *Proceedings of the International Semantic Web Conference, ISWC*, 2006.

[95] M. Weiser. The computer for the 21st century. *Scientific American*, 265(3):66–75, September 1991.

[96] M. Weiser and J. S. Brown. The coming age of calm technology, 1996.

[97] M. Weiser, R. Gold, and J. S. Brown. The origins of ubiquitous computing research at parc in the late 1980s. *IBM Systems Journal*, 38(4):693–696, 1999.

[98] D. M. Weiss. Commonality analysis: A systematic process for defining families. *Lecture Notes in Computer Science*, 1429:214–225, 1998.

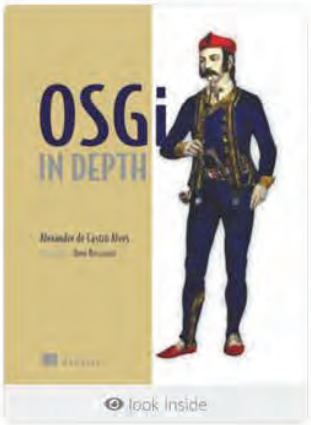
[99] D. M. Weiss and L. C.T.R. *Software Product Line Engineering*. Addison-Wesley, 1999.

[100] M. Youngblood and D. J. Cook. The mavhome architecture. Technical report, Department of Computer Science and Engineering, University of Texas at Arlington, 2004.

[101] N. Zhang, M. Ryan, and D. P. Guelev. Synthesising verified access control systems in xacml. In *Proceedings of the 2004 ACM workshop on Formal methods in security engineering*, FMSE '04, pages 56–65, New York, NY, USA, 2004. ACM.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0262

Appendix Alves 07



look inside

resources

Source Code

Book Forum

you might also like



Enterprise OSGi in Action is a hands-on guide for developers using OSGi to build enterprise Java applications. Many...

more



The Apache ActiveMQ message broker is an open source implementation of the Java Message Service spec. It makes for a reliab...

more



Spring Dynamic Modules is a flexible OSGi-based framework that makes component building a snap. With Spring DM, y...

more



OSGi is a Java-based framework for creating applications as a set of interconnected modules. OSGi lets you install, star...

more

customers also reading

This book 1° 2° 3°



OSGi in Depth ♡

Alexandre de Castro Alves

December 2011 | ISBN 9781935182177 | 392 pages
printed in black & white
Java/JVM Enterprise Software Development

eBook
pdf, ePub, online

with subscription
free or 50% off

OSGi in Depth presents practical techniques for implementing OSGi, including enterprise services such as management, configuration, event handling, and software component models. You'll learn to custom-tailor the OSGi platform, which is itself modular, and discover how to pick and choose services to create domain-specific frameworks for your business. Also, this book shows how you can use OSGi with existing JEE services, such as JNDI and JTA.

about the technology

OSGi is a mature framework for developing modular Java applications. Because of its unique architecture, you can modify, add, remove, start, and stop parts of an application without taking down the whole system. You get a lot of benefit by mastering the basics, but OSGi really pays off when you dig in a little deeper.

about the book

OSGi in Depth shows Java developers how to develop to the OSGi Service Platform Enterprise specification, an emerging Java-based technology for developing modular enterprise applications.

search inside this book

what's inside

- Deep dives into modularization, implementation decoupling, and class-loading
- Practical techniques for using JEE services
- Customizing OSGi for specific business domains

about the reader

Written for Java developers who already know the basics, *OSGi in Depth* picks up where *OSGi in Action* leaves off.

about the author

Alexandre Alves is the architect for Oracle CEP, coauthor of the WS-BPEL 2.0 specification, and a member of the steering committee of EP-TS.

table of contents

eBook
$36.00

add to cart

free with subscription

> **Gives you a deep understanding of OSGi.**
>
> David Bosschaert, Red Hat

> **Crucial if you're developing OSGi-based systems.**
>
> Steve Luetz, IBM

> **A thorough book on an increasingly important topic.**
>
> Rick Wagner, Red Hat

> **Alex has loads of experience to back up his words.**
>
> Mike Keith, Oracle

> **A jumpstart resource.**
>
> Benjamin Muschko, Weiss Inc.



table of contents

You can see this entire book for free. Click anywhere in the table of contents to start reading

detailed TOC
FOREWORD
PREFACE
ACKNOWLEDGMENTS
ABOUT THIS BOOK
ABOUT THE COVER ILLUSTRATION
1. OSGI AS A NEW PLATFORM FOR APPLICATION DEVELOPMENT
2. AN OSGI FRAMEWORK PRIMER
3. THE AUCTION APPLICATION: AN OSGI CASE STUDY
4. IN-DEPTH LOOK AT BUNDLES AND SERVICES
5. CONFIGURING OSGI APPLICATIONS
6. A WORLD OF EVENTS
7. THE PERSISTENCE BUNDLE
8. TRANSACTIONS AND CONTAINERS
9. BLENDING OSGI AND JAVA EE USING JNDI
10. REMOTE SERVICES AND THE CLOUD
11. LAUNCHING OSGI USING START LEVELS
12. MANAGING WITH JMX
13. PUTTING IT ALL TOGETHER BY EXTENDING BLUEPRINT
APPENDIX A: OSGI MANIFEST HEADERS
INDEX

about
Manning
MEAP
liveBook
liveVideo
liveProject
liveAudio
eBooks
subscriptions
tokens
our covers
geekle

info & inquiries
site reviews 54,780
user group program
write a book
create a liveProject
academic
distributors
careers
manuscript reviews
affiliate program
news

help
register pBook
placing an order
shipping & returns
why buy from Manning?
faq
contact
support

legal
privacy
terms of use
eBook license
source code
translations

© 2025 Manning Publications Co. All rights reserved.

Appendix Alves 08

https://web.archive.org/web/20120715000000*/http://www.manning.com/alves/ January 14, 2025

INTERNET ARCHIVE | WEB | TEXTS | VIDEO | AUDIO | SOFTWARE | IMAGES | SIGN UP | LOG IN | UPLOAD | Search

ABOUT | BLOG | PROJECTS | HELP | DONATE | CONTACT | JOBS | VOLUNTEER | PEOPLE

DONATE

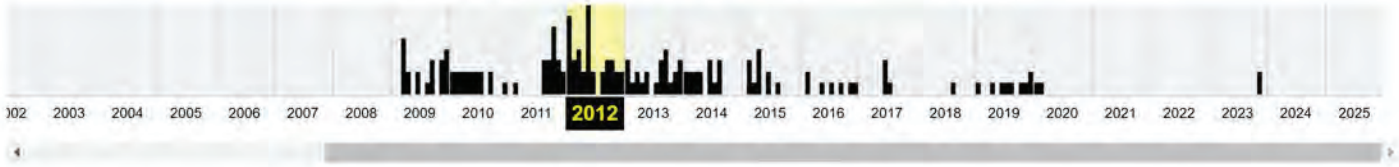
INTERNET ARCHIVE WaybackMachine

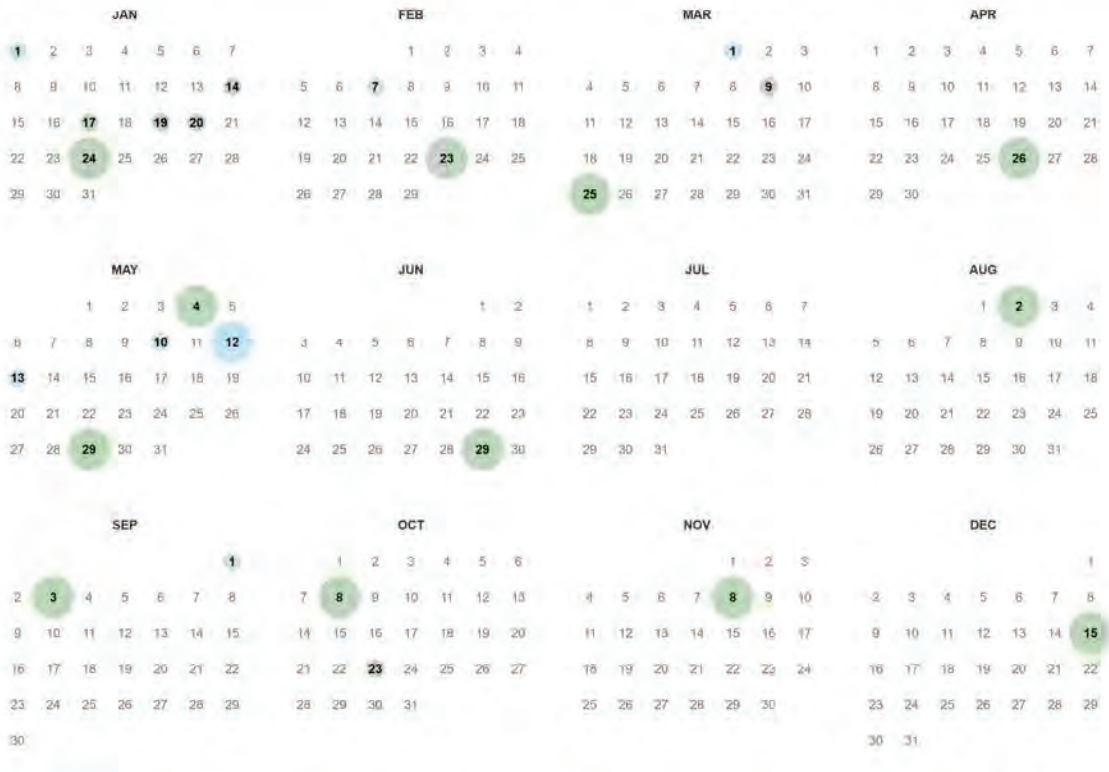
Explore more than 916 billion web pages saved over time

http://www.manning.com/alves/

Calendar · Collections · Changes · Summary · Site Map · URLs

Saved **167 times** between March 22, 2009 and November 12, 2023.





Note

This calendar view maps the number of times **http://www.manning.com/alves/** was crawled by the Wayback Machine, *not* how many times the site was actually updated. More info in the FAQ. Green indicates redirects (3xx).

FAQ | Contact Us | Terms of Service (Dec 31, 2014)

The Wayback Machine is an initiative of the Internet Archive, a 501(c)(3) non-profit building a digital library of Internet sites and other cultural artifacts in digital form. Other projects include Open Library & archive-it.org

Your use of the Wayback Machine is subject to the Internet Archive's Terms of Use

https://web.archive.org/web/20120207013932/http://www.manning.com/alves/ January 14, 2025

INTERNET ARCHIVE Wayback Machine http://www.manning.com/alves/ Go JAN FEB MAR
167 captures 22 Mar 2009 - 12 Nov 2023 2010 07 2012 2013 About this capture

Manning Publications Co.

Keep up with Manning for special offers and updates
Receive our email newsletter
Follow Manning on Twitter
Become a Manning Facebook fan



OSGi in Depth

Alexandre de Castro Alves
Foreword by David Bosschaert
December, 2011 | 392 pages
ISBN 9781935182177

ADD TO CART **$75.00** pBook + eBook (includes PDF, ePub, and Kindle)
ADD TO CART **$60.00** eBook Only (includes PDF, ePub, and Kindle)
Browse all our mobile format ebooks.

Catalog
Java
Microsoft & .NET
All by Title
All by Subject
Mobile Formats

Author Blogs
John C. Bland II
David Black
Jesus Garcia
Gojko Adzic
Felipe Leme
more...

Author Calendar
Manning Author Events

About...
Manning
Contact Us
Ordering
Support
ebooks
Covers
Sandbox Forums
Distributors
Manning Early Access Program (MEAP)
Press
Academic
User Group Program
Jobs

Search

RESOURCES

Look Inside
- Preface
- About this book
- Table of Contents
- Index
- Foreword

Resources
- Author Online
- **OSGi in Depth (Green Paper - PDF)**

Downloads
- Source Code (6.3 MB)
- Sample chapter 1
- Sample chapter 8

SUMMARY

OSGi in Depth shows Java developers how to develop to the OSGi Service Platform Enterprise specification, an emerging Java-based technology for developing modular enterprise applications.

ABOUT THE TECHNOLOGY

OSGi is a mature framework for developing modular Java applications. Because of its unique architecture, you can modify, add, remove, start, and stop parts of an application without taking down the whole system. You get a lot of benefit by mastering the basics, but OSGi really pays off when you dig in a little deeper.

ABOUT THIS BOOK

OSGi in Depth presents practical techniques for implementing OSGi, including enterprise services such as management, configuration, event handling, and software component models. You'll learn to custom-tailor the OSGi platform, which is itself modular, and discover how to pick and choose services to create domain-specific frameworks for your business. Also, this book shows how you can use OSGi with existing JEE services, such as JNDI and JTA.

Written for Java developers who already know the basics, *OSGi in Depth* picks up where OSGi in Action leaves off.

WHAT'S INSIDE

- Deep dives into modularization, implementation decoupling, and class-loading
- Practical techniques for using JEE services
- Customizing OSGi for specific business domains

ABOUT THE AUTHOR

Alexandre Alves is the architect for Oracle CEP, coauthor of the WS-BPEL 2.0 specification, and a member of the steering committee of EP-TS.

"Gives you a deep understanding of OSGi."
—From the Foreword by David Bosschaert, Red Hat

"Crucial if you're developing OSGi-based systems."
—Steve Gutz, IBM

"A thorough book on an increasingly important topic."
—Rick Wagner, Red Hat

"Alex has loads of experience to back up his words."
—Mike Keith, Oracle

"A jumpstart resource."
—Benjamin Muschko Webs Inc.

© 2003-2012 Manning Publications Co.

SAMPLE CHAPTER

OSGi IN DEPTH



Alexandre de Castro Alves

FOREWORD BY David Bosschaert

MANNING

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0269



OSGi in Depth

by Alexandre de Castro Alves

Chapter 1

Copyright 2012 Manning Publications

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0270

brief contents

1 ■ OSGi as a new platform for application development 1
2 ■ An OSGi framework primer 17
3 ■ The auction application: an OSGi case study 54
4 ■ In-depth look at bundles and services 93
5 ■ Configuring OSGi applications 131
6 ■ A world of events 161
7 ■ The persistence bundle 189
8 ■ Transactions and containers 205
9 ■ Blending OSGi and Java EE using JNDI 222
10 ■ Remote services and the cloud 249
11 ■ Launching OSGi using start levels 270
12 ■ Managing with JMX 297
13 ■ Putting it all together by extending Blueprint 316

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0271

1 OSGi as a new platform for application development

This chapter covers

- Underlying concepts of development platforms
- OSGi technology, including the framework and the enterprise services
- The benefits of using OSGi for the development of enterprise-grade applications
- The relation of Enterprise OSGi to Java Standard Edition and Java Enterprise Edition
- The current OSGi players in the market

We've all used development platforms in the past, such as Java Enterprise Edition (JEE), and even though there have been great advances in this industry, we're still building large complex systems, which are hard to develop, maintain, and extend.

OSGi provides a new development platform, based on modular decoupled components and a pluggable dynamic service model. In this book, you'll learn that OSGi is the ideal platform for the development of full-fledged, enterprise-grade, maintainable applications. Furthermore, we'll look in depth at how OSGi applications

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0272

can use a plethora of carrier-grade infrastructure services, such as HTTP, configuration, deployment, event handling, transactions, persistence, RMI, naming and directory services, and management.

We'll start this chapter by exploring development platforms and the benefits of using such platforms to develop software. We'll then discuss the requirements of a platform intended for the development of enterprise-grade applications. Next, we'll focus on the OSGi technology, expanding into its core pieces, called the OSGi framework, and its enterprise services. Finally, you'll learn why OSGi is a good fit as a development platform, particularly in light of existing solutions, such as JEE. Let's start by examining the basics.

1.1 What are development platforms and application frameworks?

In the context of software development, a *development platform* is a set of software libraries and tools that aid in the development of software components, and the corresponding runtime environment that can host these developed components, as shown in figure 1.1.

The *runtime environment* may consist of the hardware, operating system (OS), and supporting runtime libraries. One example of a runtime environment is the Java Runtime Environment (JRE), which includes the Java virtual machine (JVM) that isolates the developer from the details of the underlying OS and hardware.

Software frameworks are specialized types of a development platform's libraries and tools. Wikipedia defines a software framework as an "abstraction providing generic functionality that can be selectively specialized to provide specific functionality."

Particularly interesting to us are application frameworks. An application framework is a type of software framework whose purpose is to provide a structure for the creation of software applications. Applications are programs that allow users to perform related tasks together. Examples of software applications are document editors and antivirus software.

Putting it all together, a development platform allows a developer to create applications and to host these applications so that end users can use them. Throughout this book, it's important to keep these two players in mind: the developer (you) and

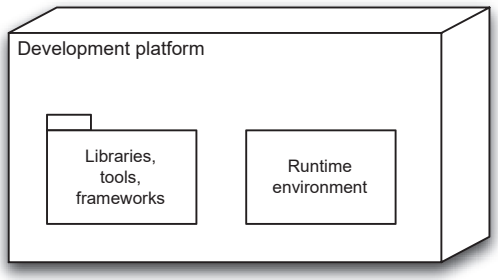


Figure 1.1 A development platform consists of a software framework and its supporting runtime environment.

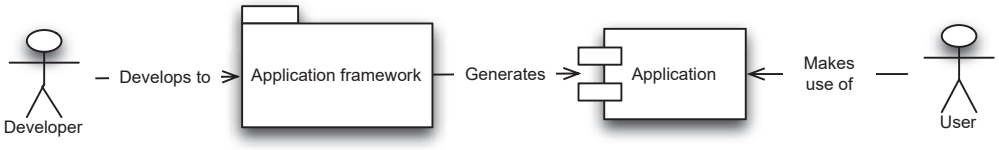


Figure 1.2 A developer creates an application, which is used by a user, and develops to a framework.

the end user, as illustrated in figure 1.2. Software development platforms are also called toolkits or SDKs (software development kits).

Historically, development platforms have always played an import role in software. The Java platform, also known as Java Standard Edition (JSE), is one example. In this case, the Java development kit (JDK) provides the software framework, and the Java Runtime Environment (JRE) provides the runtime environment. The OSGi Service Platform, which is the subject of this book, is another example of a development platform. The OSGi Service Platform uses the JRE as its runtime environment. In addition, it provides an application framework layered on top of the JDK. We'll look into its details throughout this book, but first let's see why it's important to use a development platform to begin with.

1.1.1 Why use a development platform?

Consider the following definition of the word *framework*:

> *"an essential supporting structure of a building, vehicle, or object"*

Why is the supporting structure of a vehicle important? It clearly sounds like it's important, but let's see if I can articulate why that's the case. I can think of two main reasons:

- It guarantees that I'm sitting on top of something that's solid—something that has been designed properly, implemented suitably, and tested thoroughly. A framework helps to decrease defects. This is a runtime characteristic.
- It gives the manufacturer an opportunity to reuse the frame for different vehicles. A framework helps to improve productivity through reuse. This is a design-time characteristic.

It's no different for development platforms. Development frameworks allow the creation of new applications in a form that's both efficient and has a high degree of quality.

1.1.2 Enterprise platforms

Enterprise platforms are development platforms that support the creation of enterprise applications—applications that implement business processes, business logic, or business integration to an enterprise. Examples of enterprise applications are a loan-approval application, an order-processing application, a customer relationship management (CRM) application, and a travel management application.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0274

The following two aspects characterize enterprise platforms:

- Enterprise platforms provide a collection of infrastructure-level utilities and services common to many businesses and industries, such as management, directory service, monitoring, and distribution.
- Enterprise platforms must scale, perform efficiently, and be robust and fault tolerant.

Following up on our previous example, Java also has an enterprise version, which is called Java Enterprise Edition (JEE). Other examples of enterprise platforms are Microsoft's .NET Framework, SpringSource's tc Server Development Edition, and to some extent, Google's Web Toolkit. Recently, a new enterprise platform has been developed, the OSGi Service Platform Enterprise Specification.

As you can see by the number of players, enterprise technology is quite mature, so why is there a need for a new platform, such as the one being provided by OSGi? We'll address this question in section 1.3, but before we can do that, you need to understand OSGi a bit better.

1.2 The OSGi technology

The Open Service Gateway initiative (OSGi) was formed in March 1999 by a consortium of leading technology companies with the mission to define a *universal integration platform for the interoperability of applications and services.*

When I first read their mission, it gave me the impression of being both overly complex and somewhat outdated. Hadn't people already created a universal platform for applications? As you'll learn, no one has been able to do it successfully.

1.2.1 The problem domain

First, let's investigate the underlying problem that these companies were facing. The initial members of the OSGi alliance were in a large part telecommunication equipment manufacturers and service providers. They were interested in deploying software applications on small-memory devices. For example, consider a mobile phone as the device and a location-tracking application and an advertisement application as the software applications being deployed to the mobile phone. The location-tracking application uses the mobile phone to verify the current location of the subscriber and informs the advertisement application of the location. The app then retrieves selected advertisements that are suitable to the current location of the subscriber, such as promotions from nearby restaurants, as shown in figure 1.3.

This seemingly simple interaction caused the equipment and service providers several interesting problems. First, the devices tend to have different hardware and thus different programming APIs. Hence, each vendor had to program its applications to a specific device and then port to other devices. Second, not only do the various hardware devices use different programming APIs, but there's also a large variation in their functions and capabilities. Some have more memory than others; some have a disk whereas others are completely diskless; some have GPS and some do not. Third, the

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0275

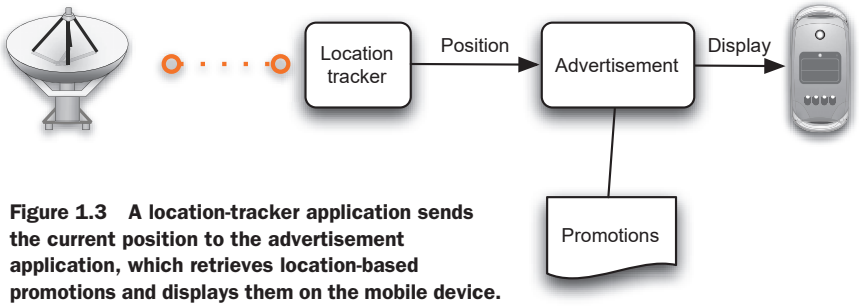


Figure 1.3 A location-tracker application sends the current position to the advertisement application, which retrieves location-based promotions and displays them on the mobile device.

lifetime of these devices is generally between one to two years, which means that new applications are likely to be created during this period, partly because of changing market demands. These new applications need to be dynamically deployed to the devices and join the existing collaborating applications that are currently running in the devices. And finally, because of the scarcity of the resources, these applications needed to closely cooperate with each other in a concise and, more important, lightweight manner.

This is no simple matter after all. Is there an existing universal platform that could help us, or does one need to be created?

Again, let's go through the problems.

PROBLEM: COPING WITH DIVERSE PROGRAMMING APIS

The first problem is portability. We need a single programming platform that abstracts the application from the underlying operating system and the hardware. In other words, we need a virtual machine. Does anything come to your mind? Yes, of course. Let's use Java to solve our first problem.

PROBLEM: VARYING DEVICE CAPABILITY

The second problem is subtler. Let's consider a specific case. The advertisement application can retrieve the available promotions from different sources. If the device has ample bandwidth, the data source could be remote. If the device doesn't have enough bandwidth but has a disk, then the promotions could be retrieved in the background and cached in the local disk as the subscriber enters a location, as shown in figure 1.4.

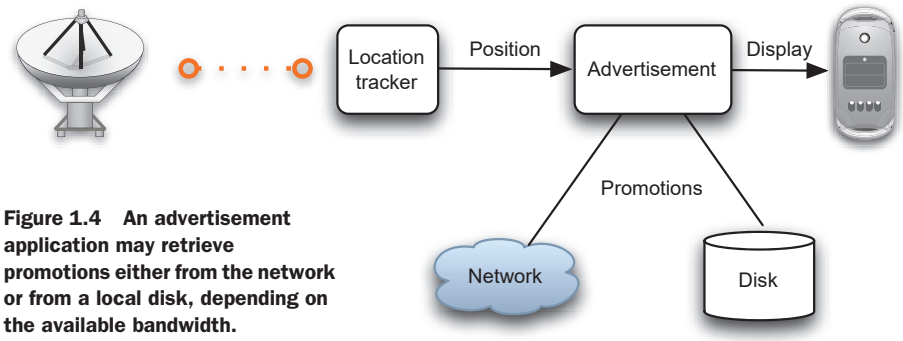


Figure 1.4 An advertisement application may retrieve promotions either from the network or from a local disk, depending on the available bandwidth.

There's a clear service contract between the advertisement application and the data source, but the implementation of this contract will vary depending on the device. This universal platform must make it easy for applications to decouple service contracts from the service implementation. The Standard Edition of Java (JSE) doesn't have a service registry or a service management framework that could help us with this. Hence, this facility either needs to be implemented from scratch, or we could try to borrow something from the Enterprise Edition (JEE) of Java. Let's hold on to this thought and tackle it after going through the other problems.

PROBLEM: SUPPORTING DYNAMIC CHANGES

The third problem can be summarized by the following requirement: the platform must allow the dynamic deployment and undeployment of applications in a secure form. Does JSE have support for this? Not really. You could try to solve this with the Java class loaders, but it wouldn't be easy, and there's no simple way of unloading classes after they've been loaded, not to mention that there's no concept of an application deployment unit. The closest concept to an application deployment unit is the idea of JARs (Java Archives), but JARs by themselves don't provide all the metadata that's needed, such as a unique naming schema for the applications.

As with the previous problem, we can implement our own solution for dynamic deployment of applications or try to leverage something from JEE. For example, web servers do have the concept of web applications, which are defined as part of a WAR (Web Archive) deployment unit file.

PROBLEM: PROVIDING A LIGHTWEIGHT SYSTEM

This brings us to the last problem. Whatever solution we pick, it must be lightweight. Yes, we could try to leverage a directory service such as JNDI from JEE or leverage the architecture from web servers, but these solutions would fail to consider the size and memory constraints enforced by the devices onto the platform, making it less suitable for embedded solutions and not a viable option.

1.2.2 The solution: a dynamic module system for Java

Java addresses some of the problems we've discussed, such as portability, but not all of them. For instance, it lacks proper support for dynamic service management.

Enter the OSGi Service Platform. In its most succinct definition, the OSGi Service Platform, or OSGi platform for short, is a dynamic module system for Java. In OSGi terminology, a Java module is called a *bundle*.

The OSGi Service Platform is composed of two main components, the OSGi framework and the OSGi services, as shown in figure 1.5.

THE OSGI FRAMEWORK

The OSGi framework provides its users with all the pieces that we discussed in the previous section:

- A portable and secure execution environment based on Java
- A service management system, which can be used to register and share services across bundles and decouple service providers from service consumers

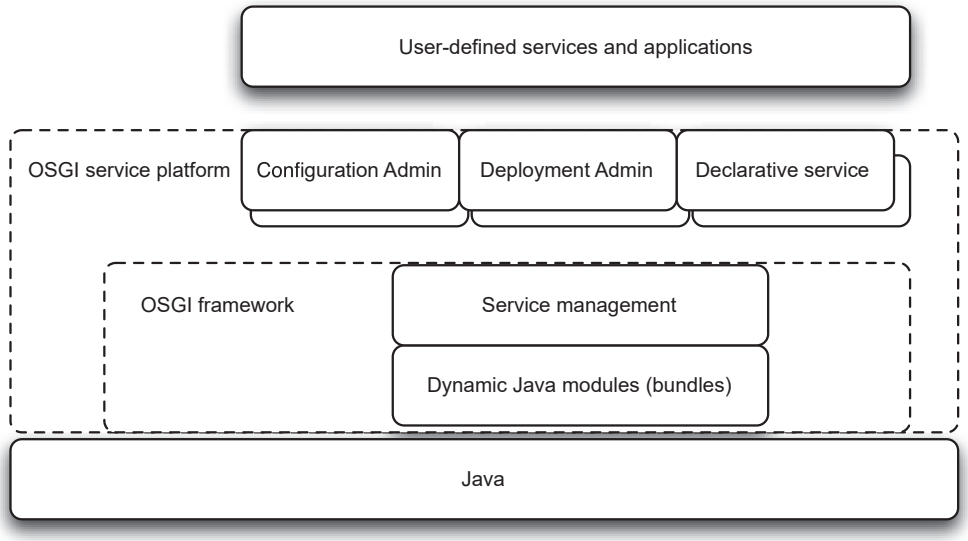


Figure 1.5 The OSGi Service Platform comprises the OSGi framework and the OSGI services.

- A dynamic module system, which can be used to dynamically install and uninstall Java modules, which OSGi calls bundles
- A lightweight and scalable solution

The OSGi framework is the core structure of the OSGi Service Platform. It can be seen as a backplane that hosts bundles, possibly containing services. If you consider that a bundle can be an application, then the definition of the OSGi framework is in accordance with our definition of application frameworks. That is to say, the OSGi framework is an example of an application framework.

Right now, we'll leave the definition of the OSGi framework somewhat loose. Let's not worry about what exactly bundles and services are. We'll discuss the framework, its concepts, and its APIs in detail in the next chapters.

THE OSGI SERVICES

Alongside the OSGI framework, the OSGI Service Platform includes several general-purpose services. You can think of these services as native applications of the OSGi Service Platform.

Some of these services are horizontal functions that are mostly always needed, such as a logging service and a configuration service.

Some are protocol related, such as an HTTP service, which could be used by a web-based application.

And finally, some services are intrinsically tied to the framework, which won't work without them. Examples of these are the bundle wiring, which manages the dynamic module system itself, and the start-level service, which manages the bootstrap process of the framework.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0278

The focus of the initial releases of the OSGi Service Platform had been on the OSGi framework, but gradually we see the OSGi services playing a more prominent role. This trend toward other components, such as services, built on top of the core OSGi framework is a reflection of the increasing popularity of the technology.

1.2.3 The Enterprise OSGi

As can be deduced from its history, OSGi was initially employed in the embedded market. But with its growing popularity and maturity, OSGi is moving to the enterprise market. To address this requirement, the OSGi Enterprise Expert Group (EEG) created the OSGi Service Platform Enterprise Specification (Enterprise OSGi).

This specification combines OSGi services that can selectively be used to provide enterprise functionality. These services can be grouped into enterprise features. Examples of enterprise features, as shown in figure 1.6, are the following:

- *Management and configuration*—This group includes the Configuration Admin service as well as the JMX Management Model service and the Metatype service.
- *Distribution*—This feature allows the communication of end points between remote instances of OSGi framework instances. Some of the services in this group are the Remote Service and the SCA Configuration Type.
- *Data access*—This feature includes support for JDBC, JPA, and JTA and allows the manipulation of persisted objects.

Before getting into the details of the OSGi technology itself, you should understand the high-level benefits of using OSGi as your development platform. We address this topic in the next section.

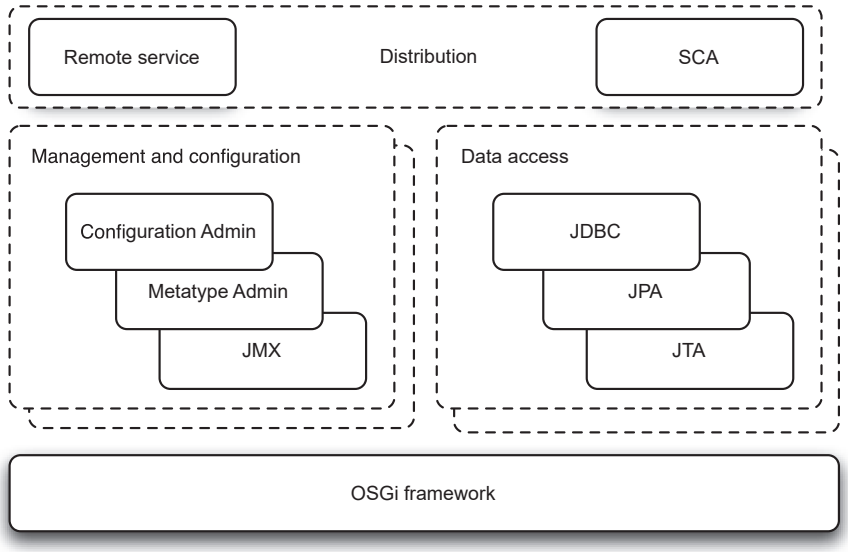


Figure 1.6 Enterprise OSGi consists of the OSGi framework and several OSGi services; together they provide enterprise features.

1.3 Benefits of using the OSGi platform

Why is the OSGi technology a good development platform for Java applications? Even further to the point, why or when is it better than the existing platforms on the market? To answer this question, we need to consider the problems we face when we develop full-fledged carrier-grade applications today.

For simplification, we can categorize these problems into two groups: problems intrinsic to the development of complex applications and problems related to existing development platforms. The problems intrinsic to applications include the following:

- As applications become larger and more complex, they become harder to maintain, sometimes exponentially so!
- Applications are difficult to extend without causing their erosion.

The problems related to existing platforms include these:

- Existing platforms are large, heavyweight systems and thus are complex to learn and use.
- There's a lack of portability among software vendors, making it difficult to reuse or share vendor components, even at the API level.

We'll explore each of these problems individually in the next sections. If the OSGi technology is able to help us address the problems of both of these categories, then not only is it suitable for the development of applications, but it's also a better tool for doing so.

1.3.1 OSGi manages the complexity of large systems

As developers, we've one time or another all faced the problem of complexity. Things are good when we're working by ourselves, on a separate, isolated piece of code. But as the team grows, from one person to ten, and the code grows from a few thousand lines to several hundred thousand, so does the complexity of working with the code and the team, and the bad news is that the increase isn't linear.

I'm positive all of the following will sound familiar:

- A simple change to the implementation of one component causes breakages throughout the application, at apparently dissociated locations.
- No one on the development team knows with certainty whether an interface can be changed without breaking existing clients.
- There are several closely related versions of the same utility functions in the application's source code.

How is the problem of managing large systems related to enterprise applications? Enterprise applications are by their very nature complex systems because of the following two factors:

- Business processes and business logic are inherently complex.

- Enterprise applications need to deal with complex issues such as resilience, management, and distribution.

How can OSGi help you manage the complexity of large systems? OSGi decreases complexity by allowing you to efficiently modularize your code and thus deal with smaller problems one at a time. By designing your code as independent modules that interact collectively to achieve the application's goal, rather than as a single monolithic structure, you're able to apply the millennia-old strategy of *divide and conquer* to your solutions.

Remember that the OSGi framework allows you to define Java modules, or bundles. These bundles have formal versioned interfaces, which must be explicitly referenced by any consuming client. In fact, by defining formal contracts between producers and consumers of code, you're able to decrease the likelihood of experiencing the three problems raised in the preceding paragraphs.

For example, let's look at the first problem again in detail. Consider bundle `B`, which contains three packages: `p`, `q`, and `r`. Packages `p` and `q` contain only implementation classes and don't need to be public, whereas `r` contains public user interfaces, as shown in figure 1.7. There's a bug in class `C` of package `p` (`p.C`) that needs to be fixed. If you were using the OSGi framework, you could have specified that the packages `p` and `q` of bundle `B` are not public. This means that the only code that has visibility to these packages would be within the bundle itself, which would allow you to restrict testing to the bundle and to any consumers of the public package `r` when `p.C` is changed. By modularizing your code, you have better control over it and know what the impact will be when something changes.

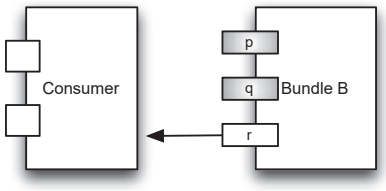


Figure 1.7 Bundle `B` with private packages `p` and `q` and public package `r`, which is being used by a consumer

In this particular case, could you have achieved the same results by meticulously coding and making sure that all Java classes are final, using the least open accessibility modifier (for example, private members), and so on? Perhaps, but would it be efficient or even possible to do these tasks on a large scale, involving several people and thousands of lines of code? Most certainly, it would not.

Furthermore, keep in mind that in OSGi the contract between producers and consumers is specified declaratively, that is, not in Java. This gives you enormous potential for tooling. For example, you could find out the transitive closure of all classes that should be tested when a class is changed.

Nevertheless, as brilliantly stated by Fred Brooks in 1986, there are no silver bullets. It's still the developer's responsibility to design adequately. For example, there'd be no point in using the OSGi framework to achieve modularization and then make everything public. We'll address modularity in the following chapters, so don't worry if the details aren't clear yet. The main point to understand is that the OSGi framework improves modularity, which in turn decreases the complexity of managing large projects and increases reuse.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0281

1.3.2 OSGi provides extensibility without eroding the system

Successful applications commonly need to be extended throughout their lifecycle; this is largely driven by changes to business requirements in today's fast-paced markets.

The problem with extensions is that they open up your system. Extensions are like public APIs, but they're more problematic because people have greater flexibility with extensions than with public APIs. You'll find that people sometimes do the unexpected with extensions.

Extending a system slowly helps erode it.

This is similar to software maintenance. As software ages, fixing bugs becomes harder and harder. Every code change takes longer to make and has a greater potential of causing other problems in the software. The reason for this erosion is that both when adding extensions and fixing bugs you're incorporating new code that wasn't made by the original authors of the software.

How can you restrain the erosion? You have to bind and control the new code. How can OSGi help you with binding and controlling extensions? As you've seen, OSGi defines the concepts of services, service consumers, and service providers.

A service consists of an interface and an implementation. The service consumer only sees and uses the service interfaces, whereas the service provider supplies the service implementations and doesn't interact directly with the consumers.

Generally, extensions of an application framework play the role of service providers, and the actual framework plays the role of the service consumer, as shown in figure 1.8. The service interface defines the contract of the extension; in other words, it's the extension point.

In this case the extension hooks into the lifecycle of the framework, and the framework calls back into the extension when appropriate. One example is an extension that wants to be notified when events of a certain type are received by the framework.

Sometimes extensions also act as the service consumers. The framework still defines the service interface, but it also provides the service implementation, which is then used by extensions.

Regardless of the approach, by keeping the extensions decoupled from the framework as separate service providers or consumers and by having a formal extension contract, you're able to isolate the extension code and thus decrease the overall erosion of the system.

Furthermore, OSGi allows you to dynamically manage the service providers. For example, OSGi supports the shutdown of a service provider that might be misbehaving without impacting the rest of the system.

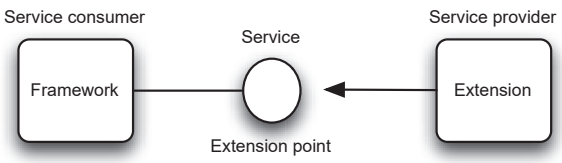


Figure 1.8 Extensions provide services through extension points. The framework consumes the services.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0282

1.3.3 OSGi is lightweight and customizable

As you've seen, the applications you develop can become quite complex. This complexity has historically also been reflected in the development platforms. To support the complexity of full-fledged applications, the development platforms comprise collections of features, APIs, and tools and therefore have become heavyweight and complex themselves. They try to be a one-size-fits-all solution to all the requirements of all businesses.

For example, a loan-approval application may need to interact with a credit-checking system using web services, whereas an order-processing application interacts with its partner using some messaging middleware, such as JMS. Regardless of these different requirements, the existing development platforms include both web services and JMS technologies for both applications' runtimes. Even though the order-processing application uses only JMS, its runtime also ends up paying the price of a web services stack.

This may seem inconsequential initially, but consider that there are dozens of different enterprise technologies, as you've seen in the OSGi services section, and thus a simple hello world application may end up having a runtime that takes megabytes of memory and seconds to minutes to start. No matter what, this complexity leaks out to us developers in different forms. Our iterative development lifecycle becomes slow, we must learn more APIs than we need to use, the programming model becomes complex, and so on. We can all relate to how complicated it is to deploy the simplest of applications to any enterprise platform today.

The OSGi platform addresses this by providing a bare-bones framework, the OSGi framework, to which services can be added a la carte. For example, if you need a web services stack, you can install it; otherwise, it's not present. The OSGi platform can be customized to be as lightweight or as complex as needed. Furthermore, this flexibility shows in different ways; for example, being able to install features dynamically means that the lifecycle of new features provided by software vendors can be shortened. You don't need to wait a year or two for the next version of your application servers; instead, you can download and install new enterprise features by themselves as soon as they become available.

1.3.4 OSGi allows for portability

Java Enterprise Edition and some of the other enterprise platforms are standardized. This means that in theory you should be able to move a JEE application from one JEE application server to another, albeit hinging on the fact that you must use only standard APIs and no vendor extensions. This capability of being able to host an application on a different vendor's application server is called *application portability*.

Yet, there's another level of portability, not always mentioned, which is that of vendors' features themselves. For example, wouldn't it be nice to be able to use a vendor's JMS implementation with another vendor's web services stack in a single container? Why would you want to do this? Among other reasons, here are the three main ones:

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0283

- It allows you to pick and choose the best-of-breed implementations of different features across all vendors. For example, one vendor might be known for its messaging implementation, whereas another might have more experience with persistence service.
- It allows you to make use of new features that may be available only from certain vendors.
- Being able to move a particular feature to a different container means that you can use vendor extensions and still achieve application portability, because you can migrate the container's features alongside the application.

So, whereas most enterprise platforms try to standardize their entire APIs as a single unit, OSGi standardizes the APIs piecemeal, in modules, allowing the vendors to provide smaller pieces and the developers to select and reuse the pieces they find to be better.

OSGi also has another major advantage over other standards such as the JEE specification: simplicity. As you'll see, you can create an OSGi-compliant Java module by adding a few lines to a Java's MANIFEST.MF file. Conversely, a JEE-compliant module needs several JEE configuration files, annotations, and Java classes that implement technology-specific Java interfaces. Simplicity plays well with standardization. Vendors are more apt to invest in standardization if the cost isn't prohibitive. Take a look at the Apache Software Foundation projects at http://www.apache.org/. You'll notice that several projects, such as Derby, have already been made into OSGi modules. Would this have happened if the process of making a library into an OSGi module was costly? I doubt it.

So far we've looked at four benefits of using OSGi. In the next section, we'll sum these up in one practical example.

1.4 Building blocks: the essence of OSGi

Let's reconsider the location-specific advertisement application of section 1.2.1. There are several features or services you could add to it:

- You could persist the selections of the promotions preferred by the mobile client, which later could be used for data mining.
- You could inform the correspondent of the promotion (for example, the store) that the mobile client has received the promotion and spent more than some considerable amount of time looking at it, perhaps through the use of an event-dispatching service.
- You could provide a mechanism that allows configuring of the application.

Figure 1.9 depicts the advertisement application and the services it uses.

Put simply, the most important thing to learn in this chapter is that the OSGi platform provides you with the means to quickly, efficiently, and easily build applications just by putting together *building blocks*. OSGi allows you to use and integrate building blocks that are being provided by different vendors, unknown to them. As you'll see,

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0284

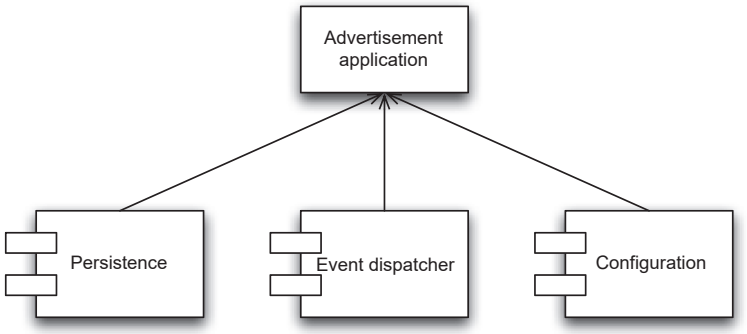


Figure 1.9 Application makes use of enterprise-based building blocks

you can even replace a building block dynamically, if a newer, better version is available from the same vendor or from a different vendor.

In this book, you'll learn how to use OSGi to create modular enterprise-grade applications by reusing infrastructure-based building blocks. You'll also learn how to perform the role of a building block vendor, providing your own reusable infrastructure services.

Collectively, all of these benefits have triggered the dissemination of public libraries of features—repositories of OSGi modules. We'll go through a list of such repositories in the next section.

1.5 Players

A multitude of vendors have implemented some aspect of the OSGi Service Platform, be it just the framework or selected services. This is no doubt a testament to OSGi's simplicity and its modularization.

Some of the most commonly used implementations of OSGi frameworks are these:

- Eclipse Equinox: http://www.eclipse.org/equinox/
- Apache Felix: http://felix.apache.org/site/index.html
- Knopflerfish: http://www.knopflerfish.org/

Most of these framework vendors also publish a public repository of services. For example, the following URLs point to these different vendors' repositories:

- Felix Repository: http://felix.apache.org/obr/releases.xml
- Knopflerfish Repository: http://www.knopflerfish.org/repo/repository.xml

In addition, several new projects have been created to tackle the implementation of the Enterprise OSGi. The goal is to allow developers to build enterprise applications using standard Enterprise OSGi services and the framework implementation of their choosing. Following are two projects that are working on implementing the Enterprise OSGi:

- Eclipse Gemini: http://www.eclipse.org/proposals/gemini/
- Apache Aries: http://incubator.apache.org/aries/

Finally, diverse development platforms, which have distinct goals, are using OSGi internally to implement their solutions. Some of these completely hide OSGi, whereas some do actually expose OSGi in some form or other. For example, a common exposure is to treat the OSGi module as the deployment unit of their platforms. Regardless, here are some examples of development platforms that are OSGi based:

- IBM WebSphere Application Server
- Oracle (formerly Sun) GlassFish Application Server
- Eclipse Virgo (SpringSource dm Server)
- JBoss Application Server
- Apache Camel
- Apache Sling
- Apache ServiceMix
- Apache Karaf

This is a reasonable number, especially if you consider that OSGi is still an emerging technology. Furthermore, because you can mix and match the components from different vendors, the overall gains are even larger! For example, you can use the Felix OSGi framework with the Eclipse service implementations and perhaps implement a few services of your own along the way.

By now, you're probably convinced and excited to use OSGi, but does this mean you need to start from scratch? We'll tackle this issue in the next section.

1.6 Are we starting from scratch?

If you're going to develop to the OSGi platform, does it mean that you need to drop your existing development platforms, such as JEE, and learn a new technology from scratch yet again? Or, even more important, can you leverage code from any of your existing enterprise applications?

Fortunately, the OSGi platform builds heavily upon JEE. As you'll see, there'll be several cases where we use, in some form or other, existing specifications defined by JEE, just in a slightly more modularized and isolated manner. This means that you're able to use the skill sets you learned in the past.

In addition, because existing JEE applications have been coded to APIs that also exist in the OSGi platform, you can move over some of the code from JEE containers to OSGi containers; however, this isn't always possible, and even when it is possible, some rewriting may be needed.

Finally, the reverse is also true; you should be able to migrate OSGi applications to JEE in some form. Furthermore, there's an even more appealing option in this case, where OSGi applications can be hosted in existing JEE application servers. This is done by hosting the full OSGi framework on top of JEE, and it's possible because OSGi is, after all, lightweight and modular.

As you'll see, the benefits of using OSGi are numerous, but in the end it all hinges on the fact that OSGi allows you to modularize both the static as well as the dynamic

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0286

structure of a program in a productive and efficient form. You may point out that the concept of modularization has been used in software engineering for decades, so why is this any different now? The difference is that OSGi allows you to apply modularization in a systematic form to software systems at their very foundation, which was never attempted before.

1.7 Summary

Development platforms consist of an application framework and a supporting runtime environment. The Java platform, with the JDK and JRE, is one example of a development platform. Enterprise platforms, such as Java Enterprise Edition (JEE), add enterprise features to the platform.

The OSGi Service Platform provides a dynamic Java module system for Java. It allows Java code to be modularized and to be managed as services. The OSGi Service Platform consists of the OSGi framework and the OSGi services. The OSGi Service Platform Enterprise Specification (Enterprise OSGi) was created to support enterprise use cases. It defines a collection of OSGi services that can be used together for enterprise features.

OSGi provides the means to achieve modularization, which helps decrease and manage the complexity of large systems. OSGi provides a native extensibility mechanism through the use of services. Finally, a consortium of several large companies is driving the OSGi specifications, hence aiding with its adoption as a standard.

In the next chapter, we'll drill down into the OSGi framework in detail. In chapter 3, we'll employ what you've learned so far to build your first OSGi-powered application. But this application will lack several useful features, such as persistence. Following up on this, we'll take an in-depth look into how we can use configuration, event handling, persistence, RMI, transactions, naming and directory services, and management to develop full-fledged carrier-grade OSGi solutions.

Finally, you must keep in mind that the main purpose of OSGi is to provide you with an efficient framework for creating and integrating software building blocks.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0287

JAVA

OSGi IN DEPTH

Alexandre de Castro Alves

OSGi is a mature framework for developing modular Java applications. Because of its unique architecture, you can modify, add, remove, start, and stop parts of an application without taking down the whole system. You get a lot of benefit by mastering the basics, but OSGi really pays off when you dig in a little deeper.

OSGi in Depth presents practical techniques for implementing OSGi, including enterprise services such as management, configuration, event handling, and software component models. You'll learn to custom-tailor the OSGi platform, which is itself modular, and discover how to pick and choose services to create domain-specific frameworks for your business. Also, this book shows how you can use OSGi with existing JEE services, such as JNDI and JTA.

What's Inside

- Deep dives into modularization, implementation decoupling, and class-loading
- Practical techniques for using JEE services
- Customizing OSGi for specific business domains

Written for Java developers who already know the basics, *OSGi in Depth* picks up where *OSGi in Action* leaves off.

Alexandre Alves is the architect for Oracle CEP, coauthor of the WS-BPEL 2.0 specification, and a member of the steering committee of EP-TS.

For access to the book's forum and a free eBook for owners of this book, go to manning.com/OSGiinDepth

Free eBook
SEE INSERT

"Gives you a deep understanding of OSGi."
—From the Foreword by David Bosschaert, Red Hat

"Crucial if you're developing OSGi-based systems."
—Steve Gutz, IBM

"A thorough book on an increasingly important topic."
—Rick Wagner, Red Hat

"Alex has loads of experience to back up his words."
—Mike Keith, Oracle

"A jumpstart resource."
—Benjamin Muschko
Webs Inc.

ISBN 13: 978-1-935182-17-7
ISBN 10: 1-935182-17-X
57500
9 781935 182177

MANNING $75.00 / Can $78.99 [INCLUDING eBOOK]

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0288

Appendix Hall 01

LIBRARY OF CONGRESS CATALOG

LOGIN

LC Catalog Quick Search

SEARCH OPTIONS

Refine Your Search 1 of 1

BOOK

OSGi in action : creating modular applications in Java

Full Record MARC Tags

000		01074cam a22003017a 4500
001		16968679
005		20111027121450.0
008		110920s2011 ctua 001 0 eng
906	__	|a 7 |b cbc |c copycat |d 2 |e ncip |f 20 |g y-gencatlg
925	0_	|a acquire |b 2 shelf copies |x policy default
955	__	|b xe11 2011-09-20 z-processor to USPL |a xh00 2011-09-21 to USPL/STM |i xh10 2011-10-27 to BCCD
010	__	|a 2011283913
015	__	|a GBB0A5366 |2 bnb
016	7_	|a 015640024 |2 Uk
020	__	|a 9781933988917 (pbk.)
020	__	|a 1933988916 (pbk.)
035	__	|a (OCoLC)ocn660553379
040	__	|a UKM |c UKM |d MEU |d SINLB |d YDXCP |d BTCTA |d TOH |d DEBSZ |d DLC
042	__	|a lccopycat
050	00	|a QA76.73.J38 |b O827 2011
082	04	|a 005.2762 |2 22
245	00	|a OSGi in action : |b creating modular applications in Java / |c Richard S. Hall ... [et al.].
260	__	|a Greenwich [Conn.] : |b Manning, |c c2011.
300	__	|a xxv, 548 p. : |b ill. ; |c 24 cm.
500	__	|a Includes index.
650	_0	|a Java (Computer program language)
650	_0	|a Computer software |x Development.
700	1_	|a Hall, Richard S.

Request this Item LC Find It Item Availability



PRINT RECORD

SAVE RECORD

EMAIL RECORD

CITE RECORD

MARCXML Record
MODS Record

Report Record Errors
Ask a Librarian
Feedback

Top

Item Availability

CALL NUMBER	QA76.73.J38 O827 2011 Copy 1
Request in	Jefferson or Adams Building Reading Rooms
Status	Not Charged

LIBRARY OF CONGRESS CATALOG

LOGIN

LC Catalog Quick Search

SEARCH OPTIONS

Refine Your Search 1 of 1

BOOK

OSGi in action : creating modular applications in Java

Full Record MARC Tags

Main title
OSGi in action : creating modular applications in Java / Richard S. Hall ... [et al.].

Published/Created
Greenwich [Conn.] : Manning, c2011.

Request this Item | LC Find It | Item Availability



PRINT RECORD
SAVE RECORD
EMAIL RECORD
CITE RECORD

MARCXML Record
MODS Record

Report Record Errors
Ask a Librarian
Feedback

Top

LCCN Permalink	https://lccn.loc.gov/2011283913
Description	xxv, 548 p. : ill. ; 24 cm.
ISBN	9781933988917 (pbk.) 1933988916 (pbk.)
LC classification	QA76.73.J38 O827 2011
Related names	Hall, Richard S.
LC Subjects	Java (Computer program language) Computer software--Development.
Browse by shelf order	QA76.73.J38
Notes	Includes index.
LCCN	2011283913
Dewey class no.	005.2762
National bib no.	GBB0A5366
National bib agency no.	015640024
Other system no.	(OCoLC)ocn660553379
Type of material	Book

Item Availability

CALL NUMBER	QA76.73.J38 O827 2011 Copy 1
Request in	Jefferson or Adams Building Reading Rooms
Status	Not Charged

Appendix Hall 02

```
LEADER 00000cam  22003738a 4500
001    660553379
008    100826s2011    ctu           000 0 eng
020    9781933988917 (pbk.)
020    1933988916 (pbk.)
040    UKM|cUKM|dMEU|dSINLB|dVDXCP
049    RVEU
050  4 QA76.73.J38|bH355 2010
082 04 005.2762|222
090    QA76.73.J38|bO84 2010
245 00 OSGi in action :|bcreating modular applications in Java /
       |cRichard S. Hall, Karl Pauls, Stuart McCulloch , David
       Savage
260    Greenwich, Conn. :|bManning ;|aLondon :|bPearson Education
       [distributor],|c2011
300    xxv, 548 p. :|bill. ;|c24 cm
650  0 Java (Computer program language)
650  0 Computer software|xDevelopment
700 1  Hall, Richard S
700 1  Pauls, Karl
700 1  McCulloch, Stuart
700 1  Savage, David
980    110715|e4199|f529156
981    GOBI
```

LOCATION	CALL NO.	STATUS
Wallace Library - Offsite Collections: Books	QA76.73.J38 O84 2010	AVAILABLE

Record 1 of 3

Previous Record Next Record



Send to Phone

Permanent Link

Title **OSGi in action : creating modular applications in Java / Richard S. Hall, Karl Pauls, Stuart McCulloch , David Savage**

Publisher Greenwich, Conn. : Manning ; London : Pearson Education [distributor], 2011

LOCATION	CALL NO.	STATUS
Wallace Library - Offsite Collections: Books	QA76.73.J38 O84 2010	AVAILABLE

ISSN/ISBN 9781933988917 (pbk.)
1933988916 (pbk.)

Descript xxv, 548 p. : ill. ; 24 cm

Subject Java (Computer program language)
Computer software -- Development

Alt Author Hall, Richard S
Pauls, Karl
McCulloch, Stuart
Savage, David

More Information

Browse Shelf | Summary | About The Author
Series | You May Also Like | Tags

We may have ebooks and other electronic resources available. Look for the (e) below.

Browse Shelf



See Full Shelf

Summary

What is OSGi? Simply put, OSGi is a standardized technology that allowsdevelopers to create the highly modular Java applications that are required forenterprise development. OSGi lets you install, start, stop, update, or uninstallcomponents without taking down your entire system. The interest in OSGi basedapplications has exploded since major vendors like Sun, Spring, Oracle,BEA, and IBM (read more)

About The Author



Richard S. Hall

Richard S. Hall, a long-time committer to the Apache Felix project, works with Sun Microsystems as part of the GlassFish team.

Karl Pauls is a Felix committer.

Stuart McCulloch is a Java developer with CodeDragons, a leader in OSGi solutions, and a Felix committer.

David Savage is a Java software engineer. He is co-author of the Newton Distributed OSGi framework and Sigil tooling project, (read more)

Series



See Full Series

You May Also Like

Top Picks



Similar Series



Java Series

View Series



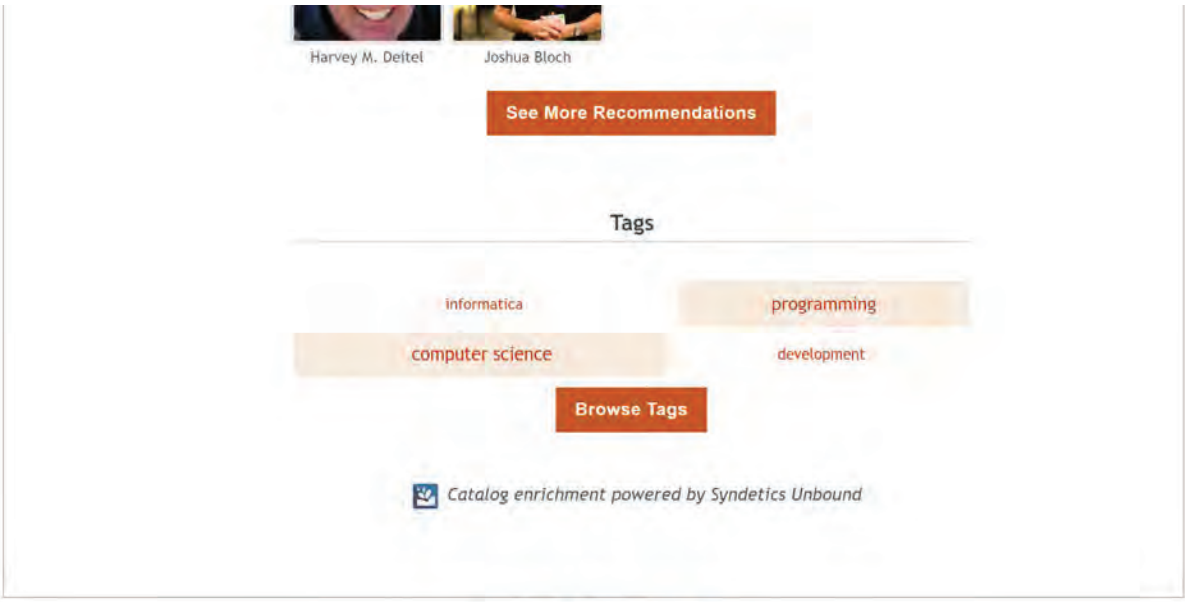
Deitel How to Program Series

Harvey M. Deitel, Paul J.

View Series

Similar Authors





Contact Us | InterLibrary Loan

Appendix Hall 03



OSGi in action : creating modular applications in Java /

Imprint:	Greenwich [Conn.] : Manning, c2011.
Description:	xxv, 548 p. : ill. ; 24 cm.
Language:	English
Subject:	Java (Computer program language) Computer software -- Development. Computer software -- Development. Java (Computer program language)
Format:	Print, Book
URL for this record:	http://pi.lib.uchicago.edu/1001/cat/bib/8403353

More details

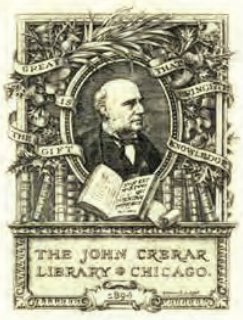
The University of Chicago Library

Purchased by the
Francis H. Dowley
Endowed Library Fund

Holdings | Description | Similar Items | Staff View

MARC

LEADER			00000cam a22000007a 4500
001			8403353
005			20130405144400.0
008			110920s2011 ctua 001 0 eng
003			ICU
010			**|a** 2011283913
015			**|a** GBB0A5366 **|2** bnb
016	7		**|a** 015640024 **|2** Uk
019			**|a** 290429900
020			**|a** 9781933988917 (pbk.)
020			**|a** 1933988916 (pbk.)
035			**|a** (OCoLC)660553379 **|z** (OCoLC)290429900
035			**|a** yb3871113
040			**|a** UKM **|b** eng **|c** UKM **|d** DLC **|d** MEU **|d** SINLB **|d** YDXCP **|d** BTCTA **|d** TOH **|d** DEBSZ **|d** JRN **|d** UKMGB **|d** CDX **|d** DEBBG **|d** OCLCO
042			**|a** lccopycat
049			**|a** CGUA
050	0	0	**|a** QA76.73.J38 **|b** O827 2011
082	0	4	**|a** 005.2762 **|2** 22
084			**|a** ST 250 **|2** rvk
245	0	0	**|a** OSGi in action : **|b** creating modular applications in Java / **|c** Richard S. Hall ... [et al.].
260			**|a** Greenwich [Conn.] : **|b** Manning, **|c** c2011.
300			**|a** xxv, 548 p. : **|b** ill. ; **|c** 24 cm.



Similar Items

Learning Apache Karaf : develop and deploy applications using the OSGI-based runtime container, Apache Karaf /
by: Edstrom, Johan
Published: (2013)

Practical domain-driven design in Enterprise Java : using Jakarta EE, Eclipse MicroProfile, Spring Boot, and the Axon Framework /
by: Nair, Vijay
Published: (2019)

Learning modular Java programming : explore the power of modular programming for building applications with Java and Spring! /
by: Jog, Tejaswini Mandar
Published: (2016)

Java EE web application primer : building Bullhorn : a messaging app with JSP, Servlets, JavaScript, Bootstrap and Oracle /
by: Wolf, Dave

336			|a text |b txt |2 rdacontent |0 http://id.loc.gov/vocabulary/contentTypes/txt
337			|a unmediated |b n |2 rdamedia |0 http://id.loc.gov/vocabulary/mediaTypes/n
338			|a volume |b nc |2 rdacarrier |0 http://id.loc.gov/vocabulary/carriers/nc
500			|a Includes index.
520			|a A guide to using the OSGi framework to create modular applications that offers step-by-step instructions for installing, starting, stopping, updating, or uninstalling modules with Eclipse plugin systems and Java EE containers, with an overview of OSGi concepts and relevant examples.
630	0	7	|a OSGi-Standard. |0 http://d-nb.info/gnd/4657143-7 |2 gnd
650		0	|a Java (Computer program language) |0 http://id.loc.gov/authorities/subjects/sh95008574
650		0	|a Computer software |x Development. |0 http://id.loc.gov/authorities/subjects/sh85029535
650		7	|a Computer software |x Development. |2 fast |0 http://id.worldcat.org/fast/fst00872537
650		7	|a Java (Computer program language) |2 fast |0 http://id.worldcat.org/fast/fst00982065
700	1		|a Hall, Richard S. |0 http://id.loc.gov/authorities/names/no2011141791 |1 http://viaf.org/viaf/163792877
903			|a HeVa
929			|a cat
999	f	f	|i a2610f21-dd8e-55e5-8aa0-8526c4641469 |s c098ed16-ca5e-593a-92ae-dbe6cb959e7d
928			|t Library of Congress classification |a QA76.73.J38 H355 2010 |l JCL |c JCL-Sci |i 1280636
927			|t Library of Congress classification |a QA76.73.J38 H355 2010 |l JCL |c JCL-Sci |e DOWL |e CRERAR |b 104155086 |i 8934325

Published: (2017)

JAVA EE 8 DEVELOPMENT WITH ECLIPSE;DEVELOP, TEST, AND TROUBLESHOOT JAVA ENTERPRISE APPLICATIONS RAPIDLY WITH ECLIPSE, 3RD EDITION.
by: RAM KULKARNI
Published: ()

THE UNIVERSITY OF CHICAGO

University Accessibility
Powered by VuFind and FOLIO

Appendix Hall 04

SEMANTIC SCHOLAR Search 221,287,909 papers from all fields of science Search Sign In Create Free Account

Corpus ID: 143430557

A Web of Things Application Architecture: Integrating the Real-World into the Web

D. Guinard • Published 2011 • Computer Science, Engineering, Environmental Science

TLDR This thesis uses the Web and its emerging technologies as the basis of a smart things application integration platform and proposes a Web of Things application architecture offering four layers that simplify the development of applications involving smart things. Expand

Share

128 Citations
Highly Influential Citations 22
Background Citations 78
Methods Citations 37
View All

View Paper [PDF] vs.inf.ethz.ch Save to Library Create Alert Cite

Figures and Tables Topics 128 Citations 211 References Related Papers

Figures and Tables from this paper



Figure 1.1



Figure 2.1



Figure 2.10



Table 2.1



Figure 2.11

View All 92 Figures & Tables

Topics

AI-Generated

SmartThings Embedded Devices Service Platforms Wireless Sensor Internet Of Things Web Of Things Application Programming Interface Pervasive Computing Barcodes

Citations

Hall Citation Type Has PDF Author More Filters Clear Sort by Relevance

Sorry, there are no results with those filters applied.

Try removing the last applied filter.

211 References

Date Range | Citation Type | Has PDF | Author | More Filters | Sort by Relevance

Information architecture for the world wide web - designing large-scale web sites

Louis B. Rosenfeld · Peter S. Morville · Computer Science · 1998

TLDR Information Architecture for the World Wide Web is a guide to how to design Web sites and intranets that support growth, management, and ease of use for Webmasters, designers, and anyone else involved in building a Web site. Expand

1,312 · 1 Excerpt · Save

Facilitating RFID Development with the Accada Prototyping Platform

C. Floerkemeier · M. Lampe · C. Roduner · Computer Science, Engineering · Fifth Annual IEEE International Conference on... · 2007

TLDR The freely available Accada RFID prototyping platform is presented and it is believed that this freely available RFID infrastructure will allow the research community to evaluate novel concepts and applications more quickly by significantly lowering the barrier for large-scale, real-world testing. Expand

45 · PDF · 3 Excerpts · Save

Mobile Interaction with the Real World: An Evaluation and Comparison of Physical Mobile Interaction Techniques

E. Rukzio · G. Broll · Karin Bee · A. Schmidt · Computer Science · European Conference on Ambient Intelligence · 2007

TLDR Context-specific user preferences are presented for the interaction techniques, to help application designers and developers decide which interaction technique to integrate into their application and which consequences this decision has. Expand

102 · PDF · 3 Excerpts · Save

Cloud Computing Research and Development Trend

Shuai Zhang · Shufen Zhang · Xuebin Chen · Xiuzhen Huo · Computer Science, Engineering · 2010 Second International Conference on Future... · 2010

TLDR Cloud computing is a computing style that provide power referenced with IT as a service that users can enjoy even he knows nothing about the technology of cloud computing and the professional knowledge in this field and the power to control it. Expand

515 · 2 Excerpts · Save

Ambient Findability: What We Find Changes Who We Become

Peter S. Morville · Computer Science, Sociology · 2005

TLDR Ambient Findability examines the convergence of information and connectivity and explores the possibilities of a world where everything can always be found--and the challenges in getting there--in this wide-ranging, thought-provoking book. Expand

185 · 3 Excerpts · Save

Security considerations in the design and peering of RFID Discovery Services

T. Burbridge · M. Harrison · Computer Science, Engineering · IEEE International Conference on RFID · 2009

TLDR This paper clearly shows that security cannot be bolted-on in the form of access control, although this is also required, and the basic communication model of the Discovery Service critically affects who shares what data with whom, and what level of trust is required between the interacting parties. Expand

11 · 2 Excerpts · Save

RFID Systems: Research Trends and Challenges

M. Bolic · D. Simplot-Ryl · I. Stojmenovic · Engineering, Computer Science · 2011

TLDR This book provides an insight into the 'hot' field of Radio Frequency Identification (RFID) Systems with an emphasis on networking aspects and research challenges related to passive Ultra High Frequency (UHF) RFID systems. Expand

209 · 2 Excerpts · Save

OSGi in Action, Creating Modular Applications in Java

Richard S. Hall · Karl Pauls · Stuart McCulloch · David Savage · Computer Science · 2011

TLDR OSGi in Action provides a clear introduction to OSGi concepts with examples that are relevant both for architects and developers, and explores numerous practical scenarios and techniques, answering questions like: How much of OSGi do you actually need? Expand

94 · 1 Excerpt · Save

From instant messaging to cloud computing, an XMPP review

A. Hornsby · R. Walsh · Computer Science · International Symposium on Consumer Electronics · 2010

TLDR The extensible Messaging and Presence Protocol is reviewed, its standardization community extensions and its usage in published literature are reviewed. Expand

49 1 Excerpt Save

Survey on Sensor Networks

Limin Wang Computer Science, Engineering 2002

TLDR This survey studies the issues and existing protocols in three areas: coverage, routing, and robustness in sensor networks, and presents several multi-path routing protocols and single-path with local repair routing protocols, which are proposed for recovering from sensor node crashes. Expand

3,432 Highly Influential 4 Excerpts Save

← ... 9 10 11 12 13 ... →

Related Papers

From the Internet of Things to the Web of Things: Resource-oriented Architecture and Best Practices

D. Guinard V. Trifa Friedemann Mattern Erik Wilde

Computer Science, Engineering · Architecting the Internet of Things

TLDR This chapter describes the Web of Things (WoT) architecture and best practices based on the RESTful principles that have already contributed to the popular success, scalability, and evolvability of the Web.

527 Save Alert

Architecting a mashable open world wide web of things

D. Guinard V. Trifa Erik Wilde

Computer Science, Engineering

TLDR This paper describes a Web of Things architecture and best-practices rooted on the RESTful principles that contributed to the popular success, scalability, and evolvability of the traditional Web and shows how Web-enabled things can be used in lightweight ad-hoc applications called "physical mashups".

104 Save Alert

Stay Connected With Semantic Scholar

Sign Up

What Is Semantic Scholar?

Semantic Scholar is a free, AI-powered research tool for scientific literature, based at Ai2.

Learn More

About
About Us
Meet the Team
Publishers
Blog
Ai2 Careers

Product
Product Overview
Semantic Reader
Scholar's Hub
Beta Program
Release Notes

API
API Overview
API Tutorials
API Documentation
API Gallery

Research
Publications
Researchers
Research Careers
Prototypes
Resources

Help
FAQ
Librarians
Tutorials
Contact

Proudly built by Ai2
Collaborators & Attributions • Terms of Service • Privacy Policy • API License Agreement

Ai2

Diss. ETH No. 19891

A Web of Things Application Architecture - Integrating the Real-World into the Web

A dissertation submitted to
ETH Zurich

for the degree of
Doctor of Science

Presented by
Dominique Guinard
M.Sc. in Computer Science, University of Fribourg
born February 27, 1981
citizen of Switzerland

accepted on the recommendation of
Prof. Dr. Friedemann Mattern, examiner, ETH Zurich
Prof. Dr. Gustavo Alonso, co-examiner, ETH Zurich
Prof. Dr. Sanjay Sarma, co-examiner, MIT Boston

2011

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0305

Abstract

A central concern in the area of pervasive computing has been the integration of digital artifacts with the physical world and vice-versa. Recent developments in the field of embedded devices have led to *smart things* increasingly populating our daily life. We define smart things as digitally enhanced physical objects and devices that have communication capabilities. Application domains are for instance wireless sensor and actuator networks in cities making them more context-aware and thus smarter. New appliances such as smart TVs, alarm clocks, fridges or digital-picture frames make our living-rooms and houses more energy efficient and our lives easier. Industries benefit from increasingly more intelligent machines and robots. Usual objects tagged with radio-tags or barcodes become linked to virtual information sources and offer new business opportunities.

As a consequence, *Internet of Things* research is exploring ways to connect smart things together and build upon these networks. To facilitate these connections, research and industry have come up over the last few years with a number of low-power network protocols. However, while getting increasingly more connected, embedded devices still form multiple, small, incompatible islands at the application layer: developing applications using them is a challenging task that requires expert knowledge of each platform. As a consequence, smart things remain hard to integrate into composite applications. To remedy this fact, several service platforms proposing an integration architecture appeared in recent years. While some of them are successfully implemented on some appliances and machines, they are, for the most part, not compatible with one another. Furthermore, their complexity and lack of well-known tools let them only reach a relatively small community of expert developers and hence their usage in applications has been rather limited.

On the other hand, the Internet is a compelling example of a scalable global network of computers that interoperate across heterogeneous hardware and software platforms. On top of the Internet, the Web illustrates well how a set of relatively simple and open standards can be used to build very flexible systems while preserving efficiency and scalability. The cross-integration and developments of composite applications on the Web, alongside with its ubiquitous availability across a broad range of devices (e.g., desktops, laptops, mobile phones, set-top boxes, gaming devices, etc.), make the Web an outstanding candidate for a universal integration platform. Web sites do not offer only pages anymore, but Application Programming Interfaces that can be used by other Web resources to create new, ad-hoc and composite applications running in the computing cloud and being

accessed by desktops or mobile computers.

In this thesis we use the Web and its emerging technologies as the basis of a smart things application integration platform. In particular, we propose a Web of Things application architecture offering four layers that simplify the development of applications involving smart things. First, we address *device accessibility* and propose implementing, on smart things, the architectural principles that are at the heart of the Web such the Representational State Transfer (REST). We extend the REST architecture by proposing and implementing a number of improvements to fit the special requirements of the physical world such as the need for domain-specific proxies or real-time communication.

In the second layer we study *findability*: In a Web populated by billions of smart things, how can we identify the devices we can interact with, the devices that provide the right service for our application? To address these issues we propose a lightweight metadata format that search engines can understand, together with a Web-oriented discovery and lookup infrastructure that leverages the particular context of smart things.

While the Web of Things fosters a rather open network of physical objects, it is very unlikely that in the future access to smart things will be open to anyone. In the third layer we propose a *sharing* infrastructure that leverages social graphs encapsulated by social networks. We demonstrate how this helps sharing smart things in a straightforward, user-friendly and personal manner, building a Social Web of Things.

Our primary goal in bringing smart things to the Web is to facilitate their integration into composite applications. Just as Web developers and tech-savvies create Web 2.0 mashups (i.e., lightweight, ad-hoc compositions of several services on the Web), they should be able to create applications involving smart things with similar ease. Thus, in the *composition* layer we introduce the *physical mashups* and propose a software platform, built as an extension of an open-source workflow engine, that offers basic constructs which can be used to build mashup editors for the Web of Things.

Finally, to test our architecture and the proposed tools, we apply them to two types of smart things. First we look at wireless sensor networks, in particular at energy and environmental monitoring sensor nodes. We evaluate the benefits of applying the proposed architecture first empirically by means of several prototypes, then quantitatively by running performance evaluations and finally qualitatively with the help several developers who used our frameworks to develop mobile and Web-based applications. Then, to better understand and evaluate how the Web of Things architecture can facilitate the development of real-world aware business applications, we study automatic identification systems and propose a framework for bringing RFID data to the Web and global RFID information systems to the cloud. We evaluate the performance of this framework and illustrate its benefits with several prototypes.

Put together, these contributions materialize into an ecosystem of building-blocks for the Web of Things: a world-wide and interoperable network of smart things on which applications can be easily built, one step closer to bridging the gap between the virtual and physical worlds.

Kurzfassung

Die Integration von digitalen Artefakten mit der physischen Welt ist ein zentrales Anliegen des Pervasive Computing. Jüngste Entwicklungen im Bereich der eingebetteten Systeme haben dazu geführt, dass wir in unserem täglichen Leben immer öfter mit "smarten" Dingen – vernetzten, digital angereicherten Geräten – interagieren. Unsere Städte werden durch drahtlose Sensor- und Aktuatornetze intelligenter und ermöglichen kontextsensitives Verhalten. Neue Geräte wie intelligente Fernseher, Wecker, Kühlschränke oder digitale Bilderrahmen machen unsere Wohnungen und Häuser energieeffizienter und unser Leben angenehmer. Die Industrie profitiert von zunehmend intelligenteren Maschinen und Robotern. Alltägliche Objekte, die mit Funkchips markiert oder mit Strichcodes versehen sind, werden um virtuelle Informationsquellen erweitert und bieten neue Geschäftsmöglichkeiten.

Als Folge dieser Entwicklung wird in den letzten Jahren im Rahmen des *Internet der Dinge* nach Möglichkeiten gesucht, smarte Dinge miteinander zu vernetzen. Um das Verbinden von Geräten zu vereinfachen, haben Forschung und Industrie eine Reihe von Niedrigenergie-Kommunkationsprotokollen konzipiert und standardisiert. Eine Folge dieser Entwicklungen war jedoch, dass sich auf der Anwendungsebene unter den verbundenen Geräten immer mehr kleine, unvereinbare Inseln bildeten. Das Erstellen von Anwendungen für smarte Dinge ist heute eine anspruchsvolle Aufgabe, die Fachwissen über jede einzelne Plattform erfordert. Dies erschwert die Integration von vernetzten Alltagsgegenständen in geräteübergreifenden Anwendungen. Um dieser Entwicklung entgegenzuwirken, erschienen immer mehr Integrationsarchitekturen die zwar zum Teil erfolgreich eingesetzt werden, jedoch meistens nicht miteinander kompatibel sind. Ihre Komplexität und der Mangel an unterstützenden Werkzeugen führen dazu, dass sie nur von einer kleinen Gruppe von Experten verwendet werden können und damit ihr Nutzen für die Erstellung von innovativen Anwendungen bisher eher begrenzt ist.

Das Internet ist ein überzeugendes Beispiel für ein skalierbares weltweites Computernetz, in dem heterogene Hardware- und Softwareplattformen ohne Integrationsprobleme zusammenarbeiten. Des Weiteren zeigt das World Wide Web, wie durch die Nutzung von vergleichbar einfachen und offenen Standards hochflexible Systeme gebaut werden können, während die Effizienz und Skalierbarkeit des Internet weiterhin gewährleistet

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0309

sind. Aufgrund seiner breiten Verfügbarkeit auf verschiedenen Geräten (z.B. PCs, Laptops, Mobiltelefone, Set-Top-Boxen, Spielkonsolen etc.) und seiner hohen Flexibilität ist das Web ein hervorragender Kandidat für eine universelle Integrationsplattform. Webseiten bieten neben ihrem klassischen Inhalt auch Programmierschnittstellen, die von anderen Web-Ressourcen verwendet werden können, um innovative Anwendungen in der Cloud zu erstellen, auf die von Desktops oder mobilen Computern aus zugegriffen werden kann.

In dieser Arbeit schlagen wir das Web of Things als eine vierschichtige Applikationsintegrationsarchitektur vor, die die Entwicklung von Anwendungen mit smarten Dingen vereinfacht. Zunächst wenden wir uns dem Problem der *Zugänglichkeit* von Geräten zu und schlagen die Umsetzung der Prinzipien, die das Herzstück des Web bilden (z.B. Representational State Transfer, REST), auf smarten Dingen vor.

Des Weiteren haben wir die REST-Architektur erweitert, um die speziellen Anforderungen der physischen Welt – etwa die Notwendigkeit für domänenspezifische Proxies oder für Echtzeit-Kommunikation – zu berücksichtigen. Ausserdem betrachten wir die Frage der *Auffindbarkeit*: Wie können in einem Netz von Milliarden von smarten Dingen diejenigen Geräte, die für eine bestimmte Anwendung benötigte Dienste anbieten, gefunden werden, und wie kann mit ihnen interagiert werden? Zur Lösung dieser Probleme schlagen wir ein leichtgewichtiges, für Suchmaschinen lesbares, Metadaten-Format vor, das mit einer Web-basierten Auffindungs- und Suchinfrastruktur zusammenarbeitet, die den besonderen Kontext von smarten Dingen berücksichtigt.

Obwohl das Web of Things ein offenes Netz von physischen Objekten unterstützt, ist es unwahrscheinlich, dass der Zugriff auf smarte Dinge in Zukunft für jedermann uneingeschränkt möglich sein wird. Aus diesem Grund bauen wir auf der dritten Ebene eine Infrastruktur, die soziale Netzwerke verwendet, um das gemeinsame Nutzen von smarten Dingen zu ermöglichen. Wir zeigen, wie ein solches *social Web of Things* die *kollektive Verwendung* von physischen Artefakten auf benutzerfreundliche Art und Weise ermöglicht.

Das Hauptziel dieser Arbeit ist die Verwendung des Web, um die Integration von smarten Dingen in kollaborativen Anwendungen zu vereinfachen. Dadurch ermöglichen wir Webentwicklern und anderen Personen mit guten Computerkenntnissen, innovative, auf smarten Alltagsdingen basierende, Anwendungen zu entwickeln – so, wie sie heute Web 2.0-Mashups (leichtgewichtige, ad hoc aus verschiedenen Web-Diensten zusammengesetzte Applikationen) erstellen. Um dieses Ziel zu erreichen, stellen wir auf einer weiteren Ebene (*Composition Layer*) das Konzept von physischen Mashups vor. Wir schlagen ausserdem eine Softwareplattform vor, die als Erweiterung eines quelloffenen Workflow-Systems konzipiert wurde und Grundkonstrukte bereitstellt, um Mashup-Umgebungen für das Web of Things zu erstellen.

Wir verwenden zwei verschiedene Arten von smarten Dingen, um unsere vorgeschlagene Architektur und die dazugehörigen Werkzeuge zu testen: Zuerst betrachten wir drahtlose Sensorknoten, insbesondere solche, die für Umwelt- und Energieüberwachung eingesetzt werden. Wir bewerten die mit dem Einsatz unserer Architektur zusammenhängenden Effekte empirisch in verschiedenen Prototypen, quantitativ über Leistungsmessungen und

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0310

qualitativ mithilfe von mehreren Entwicklern, die unsere Systeme bei der Erstellung von mobilen Applikationen und Webanwendungen einsetzen. Um ausserdem zu verstehen, wie das Web of Things die Entwicklung von intelligenten Geschäftsanwendungen vereinfachen kann, betrachten wir Systeme zur automatischen Identifizierung von Gegenständen und schlagen ein System vor, mit dem RFID-Daten in das World Wide Web und globale RFID-Informationssysteme in die Cloud integriert werden können. Wir demonstrieren das Leistungsverhalten dieses Systems anhand von verschiedenen Prototypen.

Zusammengefasst liefern unsere Beiträge ein Ökosystem von Bausteinen für ein globales Netz von interoperablen smarten Dingen, die das Erstellen von geräteübergreifenden Anwendungen vereinfachen, welche die Kluft zwischen der virtuellen und der physischen Welt überbrücken.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0311

Résumé

L'intégration du digital avec le réel reste l'une des préoccupations principales de l'informatique ubiquitaire et pervasive. De plus, les récents développements en informatique embarquée ont pour conséquence un déploiement croissant d'objets intelligents. Nous définissons les objets intelligents (appelés smart things dans cette thèse) comme des objets du monde réel doués d'une capacité de communication. Parmi les domaines d'application de ces objets, on peut citer: les réseaux de capteurs déployés dans nos villes modernes, les rendant plus intelligentes et adaptatives ou la domotique permettant à nos nouveaux téléviseurs, radio-réveils, frigidaires ou cadres à photos de nous rendre la vie plus facile et d'optimiser notre communication ou consommation d'énergie. De façon similaire, l'industrie bénéficie de robots et de machines de plus en plus intelligents et les biens de consommation sont équipés d'étiquettes électroniques ou de code-barres liés à des sources d'information virtuelles permettant de nouveaux cas d'utilisation.

Forte de l'engouement pour ces nouveau systèmes, la recherche dans le domaine de l'Internet des objets explore de nouvelles manières de connecter ces objets ensemble. Afin de faciliter ces connections, la recherche et l'industrie ont proposé plusieurs protocoles de communication. Toutefois, bien que ces protocoles facilitent la communication bas-niveau, les objets intelligents forment encore des îlots isolés les uns des autres au niveau applicatif. Par conséquent, la création d'applications pour des objets intelligents reste presque exclusivement accessible à des spécialistes du domaine de l'embarqué tout comme la création de services composites utilisant des objets intelligents. Pour contrer cette complexité, plusieurs plateformes orientées services proposent une architecture d'intégration. Toutefois, bien qu'implémentées avec succès sur quelques appareils et machines, ces plateformes ne sont souvent pas compatibles entre elles et leur complexité les rend difficiles d'accès aux novices.

En revanche, l'Internet se présente comme un très bon exemple de réseau global d'ordinateurs hétérogènes intégrés avec succès. Au-dessus de l'Internet, le Web quant à lui illustre comment un petit nombre de standards ouverts et relativement simples facilitent la construction d'applications complexes tout en préservant une certaine efficacité. De plus, la capacité du Web à supporter la création d'applications composites et interopérables ainsi que sa disponibilité pour une large palette d'appareils (p.ex., PC, portables, téléphones,

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0313

set-top boxes, consoles de jeu, etc.) en font un candidat idéal pour une plateforme d'intégration universelle. En effet, les sites Web ne sont plus de simples pages mais de véritables services qui peuvent tre réutilisés en combinaison avec d'autres sites afin de créer dynamiquement de nouvelles applications s'exécutant en ligne et dont les clients sont de natures diverses.

Dans cette thèse nous utilisons le Web et ses technologies émergentes comme base pour une plateforme applicative intégrant les objets intelligents. Au sein de l'architecture applicative du Web des objets, nous proposons quatre couches permettant de simplifier la création d'applications. La première couche traite de *l'accès aux objets intelligents*. Nous y dissertons de l'adaptation et de l'implémentation dans le monde des objets des principes architecturaux du Web. En particulier nous étudions l'utilisation du "Representational State Transfer" (REST). Nous étendons l'architecture REST en proposant certaines adaptations afin de prendre en compte les contraintes des objets intelligents. A titre d'exemple, nous étudions l'utilisation de passerelles et proposons des modèles de communication en temps réel.

Dans la deuxième couche nous étudions la *recherche et la localisation des objets*. Dans un Web peuplé de milliards d'objets, comment pouvons-nous retrouver celui fournissant le service le plus adapté à notre application? Afin de répondre à cette question nous proposons un modèle léger de méta-données que les moteurs de recherche peuvent interpréter. De plus, nous implémentons un système de registre permettant d'effectuer des recherche de services en fonction du contexte particulier des clients et des objets intelligents.

Le Web des objets tel que nous le présentons dans cette thèse promouvoit un réseau global et ouvert d'objets intelligents. Pourtant, il est peu probable que nous souhaitions laisser l'accès libre à tous nos objets au reste du monde. Dans la troisième couche nous adressons ce problème en proposant une infrastructure permettant le *partage* d'objets sur le Web. Cette infrastructure utilise les réseaux sociaux afin de permettre un protocole de partage facilement utilisable et basé sur nos connections personnelles, créant ainsi un Web social des objets.

Notre but principal lorsque nous proposons d'amener les objets au plus proche du Web est de faciliter leur utilisation dans des applications composites. Tout comme les aficionados de la technologie et du Web créent facilement des "mashups" (càd. des applications légères et dynamiques utilisant plusieurs services du Web), ils devraient pouvoir en faire de même avec les objets intelligents. En conséquence, la troisième couche traite de la composition de services et introduit la notion de *mashups physiques*. Nous y proposons une plateforme logicielle construite comme une extension d'un gestionnaire de processus et offrant des éléments de langage permettant de créer des éditeurs de mashup pour les objets intelligents.

Finalement, afin de d'évaluer l'architecture et les outils proposés, nous nous attardons sur deux types d'objets intelligents. Tout d'abord nous considérons les réseaux de capteurs environnementaux. Les bénéfices de l'utilisation du Web des objets y sont testés de façon empirique par le biais de plusieurs prototypes, de façon quantitative à l'aide d'évaluations

de performances, puis de façon qualitative par le biais d'études avec des développeurs utilisant ces approches. Ensuite, nous étudions le cas des systèmes d'identification par ondes radio (RFID) et proposons une structure permettant d'amener les données et appareils RFID sur le Web. Nous évaluons les performances de cette structure et illustrons ses bénéfices par le biais de plusieurs prototypes.

Mises ensemble, les contributions de cette thèse proposent les fondations du futur Web des objets: un réseau d'objets et de services global et interopérable au-dessus duquel des applications peuvent tre créées avec aisance. Cette thèse permet donc de diminuer le fossé qui existe encore entre notre monde de tous les jours et le monde virtuel.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0315

Acknowledgements

Earning a Ph.D. degree would be an impossible journey without the support of professors, peers and loved ones. Here, I would like to thank a few of the ones that supported me over the four years of my Ph.D. and beyond. My first and deepest gratitude goes to Prof. Friedemann Mattern. His pioneering work on the Internet of Things inspired me profoundly and his guidance, support and interest in my research were great sources of motivation.

Even if this thesis reflects my views and research, it is the outcome of a collaborative effort. Instrumental to this thesis were the fruitful collaborations with Erik Wilde at UC Berkeley and in particular with Vlad Trifa at ETH Zurich. Vlad and I shared the most creative (and often funniest!) moments. Over the years, he became one of my closest friends and his support was key to my success. Thanks Vlad!

I had the chance to split my Ph.D. time between research at ETH and SAP Research Zurich where further collaborations and friendships emerged. At ETH for example, learning the art of user studies with my office-mate and friend Iulia Ion or working with Simon Mayer and his unlimited creativity for Web of Things prototypes. I also heartily want to thank my other colleagues at ETH: Robert Adelmann, Alexander Bernauer, Christian Beckel, Philipp Bolliger, Steve Hinske, Wilhelm Kleiminger, Matthias Kovatsch, Marc Langheinrich, Benedikt Ostermaier, Matthias Ringwald, Christof Roduner, Kay Römer, Silvia Santini, Gábor Sörös and Markus Weiss. I further want to extend my gratitude to all the students who contributed, piece by piece, to building the Web of Things as we present it in this thesis: Azu Aguilar, Bettina Dober, Mathias Fischer, Soojung Hong, David Karam, Mathias Mueller, Lukas Naef and Thomas Pham.

At SAP, I would first like to thank the SOCRADES and SENSEI team-members for our collaborations: Oliver Bäcker, Markus Eurich, Stefan Haller, Stamatis Karnouskos, Moritz Köhler, Luciana Moreira Sá de Souza, Frederic Montagut, Patrik Spiess, Domnic Savio and Claudia Villalonga. I also enjoyed the very spontaneous collaborations with Ivan Delchev, Felix von Reischach and Andreas Budde. I extend my thank you to all friends and colleagues from SAP Research and especially to my manager Uli Eisert for his valuable support.

I found a significant source of inspiration for my thesis in the world of product identifica-

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0317

tion and RFID. Hence, I am thankful to Prof. Elgar Fleich and his team for letting me the chance to work as a research associate of the Auto-ID labs in Zurich for more than a year. Furthermore, I want to express my deepest gratitude to Prof. Sanjay Sarma and his team who welcomed me at the MIT Auto-ID labs. They treated me as one of theirs from day one on and for the following six months of very inspiring discussions and fruitful collaborations.

I was also lucky enough to be able to count on several mentors. In particular, Christian Floerkemeier whose guidance first at MIT and then ETH was truly valuable. His constant motivation, insights in the business of the Internet of Things and pragmatic view of research helped me to a level he probably does not suspect. Furthermore, I need to thank my two long-term mentors Olivier Liechti and Patrik Fuhrer whose guidance, sharp advices and jokes were always a source of pure motivation. I also want to thank Jacques Pasquier, Hans Gellersen and Beat Hirsbrunner for their very valuable advices and support at several important steps of my career.

Building a community while working on this thesis was a tremendous experience and I would like to heartily thank all the contributors, community members and readers of the Web of Things blog[1] as well as all the organizers, PC members and participants of the WoT workshop series[2].

Last but not least I want to express my deepest gratitude to my family: my sister and best friend Véronique, my brother-in-law Steven, and to the rest of the family for their tremendous love and support over the years. To Rachel who, thanks to her love, patience and support, was the best partner a Ph.D. student could dream of. To my mother who always believed in me and motivated me to study. To my father Jean-Pierre whose humbleness, patience, dedication and taste for all things computers were and will always be a model for me. Thank you.

[1] See www.webofthings.org
[2] See www.webofthings.org/wot

Contents

1 Introduction **1**

1.1 Motivation . . . 1

1.2 Contributions . . . 4

1.2.1 The Web of Things: A Web-Oriented Service Platform for Smart Things 4

1.2.2 Case Studies . . . 5

1.3 Thesis Outline . . . 7

2 The Web of Things **9**

2.1 Device Accessibility Layer . . . 12

2.1.1 A Web API for Smart Things . . . 13

2.1.2 Implementation Strategy: Connecting Things to the Internet . . . 22

2.1.3 Pushing Data from Smart Things and Smart Gateways . . . 26

2.1.4 Summary and Applications . . . 32

2.2 Findability Layer . . . 33

2.2.1 Search Engines and the Internet of Things . . . 33

2.2.2 A Web-Oriented Discovery and Lookup Infrastructure . . . 44

2.2.3 Evaluation . . . 53

2.2.4 Summary and Applications . . . 59

2.3 Sharing Layer . . . 60

2.3.1 Requirements for a WoT Sharing Platform . . . 60

2.3.2 Social Access Control: An Architecture for the Social Web of Things . 61

2.3.3 Retrieving the Owners' Social Graphs . . . 63

2.3.4 Registering and Sharing Smart Things and Smart Gateways . . . 66

2.3.5 Accessing Shared Smart Things . . . 68

2.3.6 Physical Feeds Aggregation . . . 70

2.3.7 Software Architecture . . . 70

2.3.8 Friends and Things: A Social WoT Web Application . . . 74

2.3.9 Summary and Applications . . . 75

2.4 Composition Layer . . . 76

2.4.1 Physical Mashups in the Web of Things . . . 76

2.4.2 From Web 2.0 Mashups Editors to Physical Mashup Editors . . . 78

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0321

2.4.3 Adapting a Web 2.0 Mashup Editor to the Web of Things 78
2.4.4 Requirements for Physical Mashup Editors 82
2.4.5 A Platform for Physical Mashups Editors 83
2.4.6 System Architecture 84
2.4.7 Discussion and Summary 89
2.5 Developers Perspectives on the WS-* Alternative Architecture 90
2.5.1 Methodology 91
2.5.2 Results 93
2.6 Discussion and Summary 98
2.7 Related Work 99
2.7.1 Device Accessibility Layer 100
2.7.2 Findability Layer 101
2.7.3 Sharing Layer 104
2.7.4 Composition Layer 105
2.8 Summary 107

3 Bringing Wireless Sensor and Actuator Networks to the Web 109
3.1 WoT General Purpose Sensing Platform 110
3.1.1 Device Accessibility Layer with End-to-End HTTP 111
3.1.2 Findability Layer 120
3.2 WoT Smart Metering 123
3.2.1 Implementing the Device Accessibility Layer 124
3.2.2 Applications 128
3.2.3 Qualitative Evaluation 130
3.3 Sharing Layer 134
3.3.1 Quantitative Evaluation 136
3.4 Composition Layer: Cross-Device Physical Mashups 138
3.4.1 The Ambient Meter 138
3.4.2 With Clickscript 140
3.4.3 Energy-Aware Mobile Mashup Editor 142
3.5 Related Work 148
3.6 Discussion and Summary 149

4 Resource-Oriented RFID Networks 151
4.1 The EPC Network in a Nutshell 153
4.1.1 Identifying EPC Numbers 155
4.1.2 Standards for Capturing EPC Events 155
4.1.3 Sharing EPC Events 156
4.2 A Cloud-Based Virtual Infrastructure for the EPC Network 156
4.2.1 Pain-Point: Complex Backend Deployment and Maintenance 157
4.2.2 Virtualization Blueprint 158
4.2.3 Cloud Computing: Utility Computing Blueprint 158
4.3 Device Accessibility Layer 159
4.3.1 Pain-Point: Complicated Applications Developments 160

4.3.2 EPCIS Webadapter . . . 160
4.3.3 Pushing from Readers to Web Clients . . . 167
4.3.4 Case-Study: EPC Find . . . 168
4.4 Sharing Layer . . . 174
4.4.1 Pain-Point: Lack of Access Control . . . 174
4.4.2 System Architecture . . . 175
4.5 Composition Layer: Auto-ID Physical Mashups . . . 177
4.5.1 Pain-Point: Tedious Business Case Modeling and Cross Systems Integration . . . 177
4.5.2 Mobile Tag Pusher . . . 177
4.5.3 The EPC Dashboard Mashup . . . 179
4.5.4 RFID Physical Mashup Editor . . . 182
4.6 Evaluating the EPCIS Webadapter . . . 185
4.7 Related Work . . . 187
4.8 Discussion and Summary . . . 189

5 Conclusions and Outlook **193**
5.1 Contributions . . . 193
5.2 Discussion and Future Challenges . . . 194

Bibliography **197**

Chapter 1

Introduction

Contents

1.1 Motivation 1
1.2 Contributions 4
1.2.1 The Web of Things: A Web-Oriented Service Platform for Smart Things 4
1.2.2 Case Studies 5
1.3 Thesis Outline 7

1.1 Motivation

Pervasive and ubiquitous computing have a long-lasting tradition of looking into integration of physical objects with the digital world. Recent developments in the field of embedded devices have led to *smart things* increasingly populating our daily life, slowly but steadily forming interconnected networks of physical objects: Sensor nodes are networked together to create environmental monitoring applications, making cities smarter and dynamically adapting to their context [196]. Home appliances such as TVs, alarm clocks, digital picture frames and Hi-Fi systems can communicate with each other to offer integrated services such as cross-devices multimedia experiences, smarter HVAC (Heating, Ventilating, and Air Conditioning) systems or more energy aware and efficient homes [113, 91, 196, 133, 134]. RFID-tagged objects in stores and along supply-chains allow manufacturers, suppliers and service providers to optimizes their operations [16, 55]. Products get digital identities through barcodes or RFID tags and offer unprecedented business opportunities [56, 57, 16, 133].

To facilitate these connections, research and industry have come up with a number of low-power physical and transport network protocols such as Zigbee, Bluetooth, IEEE 802.15.4 or, more recently, low-power WiFi and 6LoWPAN [98, 99] [244]. However, while these

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0325



Figure 1.1: Smart things are digitally enhanced objects and devices that have communication capabilities. They range from machines and home appliances to wireless sensor and actuator networks as well as tagged every-day objects.

developments help towards integrating smart things at the network layer, at the application layer, embedded devices still form multiple, small, incompatible islands: Developing applications using them remains a challenging task that requires expert knowledge of each smart things platform [36, 161]. As a consequence, smart things remain hard to integrate into composite applications.

Several service platforms propose a standardized integrated architecture to facilitate the cross-integration of smart things. Standards such as UPnP and DNLA, Jini or OSGi successfully address concerns such as service discovery and registration. However, these systems are not fully compatible with one another and their complexity and lack of well-known tools let them only reach a relatively small community of expert developers [35]. Hence their direct usage for innovative applications (e.g., mobile or Web-based applications) has been rather limited to date.

In the world of enterprise computing, interoperability and loose-coupling at the business application layer is achieved using WS-* Web services [10]. WS-* Web services, sometimes called "Big Web services" [154], are based on two main XML formalisms: WSDL and SOAP as well as a set of additional standards (WS-Addressing, WS-Security, WS-Discovery, etc.). With the goal of facilitating the integration of smart things with applications, several research initiatives look at adapting these services to the real-world [157, 36, 104, 79, 161]. This research led to lighter forms of WS-* services targeted towards real-world applications such as DPWS (Device Profile for Web Services) [104] or DNLA, both direct evolutions of UPnP.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0326

The ultimate goal of these initiatives can be summarized as trying to *create a loosely-coupled ecosystem of services for smart things*. That is, a widely distributed platform in which the services provided by smart things can be easily composed to create new applications and use-cases. As shown in several research projects, the WS-* approach is an improvement over the proprietary protocols traditionally used in this field [36, 104, 35], however the approach also has important shortcomings. First, WS-* standards are rather verbose and heavy in terms of required bandwidth, memory and CPU. This makes them challenging to implement on devices with limited resources [214, 154]. More importantly, despite their name, Web services actually use the Web as a transport layer and not as an application architecture [153] making them harder to use and integrate with the World Wide Web.

The Internet is a compelling example of a scalable global network of computers that interoperate across heterogeneous hardware and software platforms. On top of the Internet, the Web illustrates well how a set of relatively simple and open standards (e.g., HTTP, HTML, XML, JSON, etc.) can be used to build very flexible systems while preserving efficiency and scalability. The cross-integration and developments of composite applications on the Web, alongside with its ubiquitous availability across a broad range of devices (e.g., desktop computers, laptops, mobile phones, set-top boxes, gaming devices, etc.), make the Web an outstanding candidate for a universal integration platform [81].

Hence, as more and more devices are getting connected to the Internet, the next logical step is to use the World Wide Web and its associated technologies as a platform for smart things. In the *Web of Things (WoT)*, we are considering smart things as first-class citizens of the Web and position the WoT as a refinement of the Internet of Things (IoT) [133, 134, 170] by integrating smart things not only into the Internet (the network), but into the Web (the application layer).

To achieve this integration, we propose to reuse and adapt patterns commonly used for the Web. We embed Web servers [99, 42, 76] on smart things and apply the REST (Representational State Transfer) architectural style [50, 160] to the physical world. The essence of REST is to focus on creating loosely coupled services on the Web so that they can be easily reused [153]. REST is actually core to the Web and uses URIs for encapsulating and identifying services on the Web. In its Web implementation it also uses HTTP as a true application protocol. It finally decouples services from their presentation and provides mechanisms for clients to select the best possible formats. This makes REST an ideal candidate architecture to build a *universal API* for smart things. As the "client-pull" interaction model of HTTP does not fully match the needs of event-driven IoT applications, we further suggest the use of syndication techniques such as Atom and some of the recent real-time Web technologies to enable smart things push interactions.

In this thesis, we propose a Web of Things Architecture: a Web-based distributed application platform for smart things. As a consequence of the proposed architecture, smart things and their functionality get transportable URIs that one can exchange, reference on Web sites and bookmark. Smart things are also linked together enabling discovery simply by browsing. The interaction with smart things can also almost entirely happen in

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0327

a browser, a tool that is ubiquitously available and that most users are familiar with [110]. Applications can be built upon them using well-known Web languages and technologies. Furthermore, smart things can benefit from the mechanisms that made the Web scalable and successful such as caching, load-balancing, indexing and searching.

In this thesis, rather than looking at one particular challenge, we take a holistic view, looking also at the bigger picture. We propose a number of building-blocks towards creating a distributed deployment of smart things which fosters serendipitous re-use of smart things. Our goal is to create a participatory WoT where communities and users can create opportunistic applications, i.e., composite applications easily created by re-using existing services or devices. Just as people create Web mashups [90] involving Web 2.0 services, they should be able to create *Physical Mashups* [81] mixing services from the real and virtual worlds together.

1.2 Contributions

In this section we outline the main contributions of the thesis towards an Internet of Things supporting opportunistic applications or physical mashups. First, we present the Web of Things architecture. We then apply it systematically to two domains: Wireless Sensor and Actuator Networks and Auto-ID (Automatic Identification) networks.

1.2.1 The Web of Things: A Web-Oriented Service Platform for Smart Things

Our first contribution is the Web of Things Architecture. For this, we build on top of network connectivity and focus on the application layer. We take a systematic approach required to achieve the *mashability* of smart things with the Web. We identify four layers: *accessibility, findability, sharing and composition*. We study how each layer can be designed and implemented as a service on the Web using Web languages and patterns.

We begin by addressing *device accessibility*. We propose to use the REST architectural style [50] and study its applicability to smart things [81]. For this part, we build upon several projects in the field of ubiquitous computing [110, 39, 126] and look at a systematic application of the REST principles and their current Web implementation in HTTP to adapt the architecture to the needs of smart things. We then discuss two ways of integrating smart things to the Internet and the Web. Either directly or through the use of enhanced reverse proxies that we call Smart Gateways [193, 77]. As a result, smart things become easier to build upon. Popular Web languages (e.g., HTML, Python, JavaScript, PHP) can be used to easily build applications involving smart things and users can leverage well-known Web mechanisms (e.g., browsing, searching, bookmarking, caching, linking) to interact and share these devices. We illustrate this by means of a user study.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0328

We then study the *findability* [144] of smart things: once they are connected to Web, how does one find the services they offer to integrate them into composite applications? In particular, we propose a discovery and lookup infrastructure for the Web of Things and describe smart things according to a metadata model that we implement using semantic annotations based on Microformats [254]. Furthermore, we propose, implement and evaluate extending user search queries for smart things services based on a new process [70] using related keywords extracted from services on the Web (e.g., Wikipedia, Yahoo Web Search, etc.) [77].

Using smart things in composite applications requires a scalable *sharing* mechanism that lets owners of smart things manage access control in a convenient and straightforward way. We introduce the Social Access Controller [71], a platform that relies on social networks (e.g., Facebook, LinkedIn, Twitter, etc.) and their open APIs (e.g., OpenSocial) to enable owners to leverage the social structures in place for sharing smart things with others.

Finally, we discuss the *composition* of smart things on the Web. We introduce the notion of *Physical Mashups* where services from the Web are serendipitously composed with services offered by smart things. We discuss a number of requirements towards a Physical Mashups Framework upon which Physical Mashup editors can be built and propose an implementation as an open service on the Web [68].

These four layers form the basis of our Web of Things Architecture. Not every Internet of Things application that is to be ported to the Web will require the four of them, but they present a model that can be used towards a looser-coupling, better and easier integration of the physical and the virtual worlds thanks to Web technologies.

1.2.2 Case Studies

Our second and third contributions are two domains in which we apply the Web of Things Architecture in three different case studies. We first look at the field of Wireless Sensor and Actuator Networks and then at Auto-ID networks.

Bringing Wireless Sensors and Actuators Networks to the Web

In the last decade, important progress in the field of embedded systems has given birth to a myriad of tiny computers to which virtually any type of sensors/actuators can be attached. By inter-connecting these devices using low-power wireless communication, a whole new world of possible applications is unveiled. Networks of physically distributed computers, usually called Wireless Sensor Networks (WSN), are valuable tools for monitoring the physical world [196]. Likewise the same type of devices can also be used for actuating the world, such as controlling security systems, traffic lights, etc. These networks are then called Wireless Sensor and Actuator Networks[1]

[1]We will subsequently use the term Wireless Sensor Network (WSN) for both, pure sensor and sensor/actuator networks.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0329

Unfortunately, most projects using WSNs are based on different – and usually incompatible – software and hardware platforms [36, 104, 35, 6]. Within such an heterogeneous ecosystem of devices, the development of simple applications still requires special skills and a substantial amount of time [146]. Moreover, for each new deployment, a large amount of work must be devoted to re-implement basic functions and application-specific user interfaces, which is a waste of resources that could be used by developers to focus on the application logic. Ideally, developers should be able to quickly build applications only by recombining ready-made building-blocks.

Hence, the world of WSNs is an interesting case-study for the proposed approach as adopting a simple application architecture for these devices contributes to foster their wider usage and applicability. Our contribution here is to look at two WSN platforms and evaluate the validity of our approach by integrating them to the Web based on the Web of Things Architecture.

Facilitating General Purpose Sensing Applications We first look at an all purposes WSN platform called Sun SPOTs [273]. We design, implement and evaluate how each layer of the model can be leveraged to make the developments on top of this platform as accessible as simple Web development is. With this platform we also evaluate the performances when using Web protocols directly on WSNs using embedded Web servers.

Facilitating Energy Monitoring and Control Applications Rising global energy demand and the limitation of natural resources has led to increased thoughts on residential energy consumption. A necessary step towards energy conservation is to provide timely and fine-grained consumption information. This allows for users to identify energy saving opportunities and possibly adjust their behavior to conserve energy [135].

Currently available off-the-shelf products that depict the energy consumption in near real-time are helpful, but do not fully meet the user needs as they have high usage barriers and often require complex installations [60]. Furthermore, they are not able to provide the most compelling feedback [53] since they lack the ability to provide an appliance-specific break down of the energy consumption and are not able to compare the consumption of individual devices in an appealing manner. Finally, they do not meet the needs of software developers as they do not offer open APIs, and developing applications on top of them is rather cumbersome.

Hence, we propose an *easy to use, deploy and develop upon* device-level energy monitoring platform called *Energie Visible* [83, 81, 204]. It is based on an off-the-shelf smart power outlet that we seamlessly integrate into the Web by systematically applying the Web of Things Architecture. We demonstrate how this facilitates, on the one hand-side, the deployment and usage by home users and, on the other hand-side, the development of novel applications for developers. We present the design and implementation of a Web user interface. We further evaluate the suitability of our approach with the help of a pilot deployment and feedback from several developers using our framework in research

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0330

projects and prototypes such as a mobile phone energy monitoring application.

Facilitating the Development and Deployment of Distributed Auto-ID Applications

The RFID (Radio Frequency IDentification) [169, 52] standards community has developed a number of communication interfaces and software standards to provide interoperability across different RFID deployments. This extensive standards framework, known as the EPC (Electronic Product Code) Network [170], covers aspects such as reader-to-tag communication, reader configuration and monitoring, tag identifier translation, filtering and aggregation of RFID data, and persistent storage of application events. While there are in total fifteen standards that make up the EPC Network framework, the air interface protocol known as EPCglobal UHF Gen2 has seen the most adoption – both in large scale supply chain applications as well as niche RFID deployments.

The adoption of the software standards within the EPC Network has been significantly slower [172, 72]. The deployment of RFID applications that implement the EPC Network standards often remains complex and cost-intensive mostly because they typically involve the deployment of rather large and heterogeneous distributed systems. As a consequence, these systems are often only suitable for big corporations and large implementations and do not fit the limited resources of small to mid-size businesses and small scale applications both in terms of required skill-set and costs.

While there is most likely no universally available solution to these problems, the success of the Web in bringing complex, distributed and heterogeneous systems together through the use of simple design patterns appears as a viable approach to address these challenges. Thus, our contribution in this context is to study the pain points of RFID applications and systematically apply the Web of Things model [74, 72].

In particular, we show how Cloud Computing, RESTful interfaces and the real-time web as well as Physical Mashups can simplify application development, deployments and maintenance in a common RFID application. Our analysis also illustrates that RFID/EPC Network applications are an interesting fit for WoT technologies and that further research in this field can significantly contribute to making real-world applications in this domain less complex and cost-intensive.

1.3 Thesis Outline

The remainder of this thesis is structured as follow: Chapter 2 presents the Web of Things Architecture. We expose the required components for a successful Web-integration of smart things. We further propose a number of optional components that help to create a Web ecosystem in which Physical Mashups are made possible. In particular, we discuss device accessibility, findability, sharing and composition. In this part of the thesis, we also review an alternative architecture and present a user-study in which we build upon

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0331

the developers' experience to provide guidelines on making the right architectural decision for IoT projects.

In Chapter 3 we apply the architecture to the domain of Wireless Sensor Networks. In the first part we discuss the integration of a general purpose sensing platform. We then present the benefits of the Web of Things Architecture when applied to smart energy monitoring and control. Based on our open-sourcing of the platform, we also discuss a pilot deployment and illustrate the ease of use and integration that the Web of Things Architecture provides.

In Chapter 4 we show how the architecture can be applied to the Auto-ID and RFID domain. We apply it to several components of the EPC Network and illustrate, by means of prototypes and studies, how the Web of Things Architecture has the potential to simplify and foster developments in the RFID domain.

Finally, in Chapter 5 we provide an summary of our contributions. Taking a step back, we also discuss some of the open challenges and interesting future directions that we believe the Internet of Things and Web of Things domains will have to take.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0332

Chapter 2

The Web of Things

"Any sufficiently advanced technology is indistinguishable from magic"

Arthur C. Clarke

Contents

2.1 Device Accessibility Layer 12

2.1.1 A Web API for Smart Things 13

2.1.2 Implementation Strategy: Connecting Things to the Internet . . . 22

2.1.3 Pushing Data from Smart Things and Smart Gateways 26

2.1.4 Summary and Applications 32

2.2 Findability Layer 33

2.2.1 Search Engines and the Internet of Things 33

2.2.2 A Web-Oriented Discovery and Lookup Infrastructure 44

2.2.3 Evaluation 53

2.2.4 Summary and Applications 59

2.3 Sharing Layer 60

2.3.1 Requirements for a WoT Sharing Platform 60

2.3.2 Social Access Control: An Architecture for the Social Web of Things 61

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0333

2.3.3 Retrieving the Owners' Social Graphs . . . 63
2.3.4 Registering and Sharing Smart Things and Smart Gateways . . . 66
2.3.5 Accessing Shared Smart Things . . . 68
2.3.6 Physical Feeds Aggregation . . . 70
2.3.7 Software Architecture . . . 70
2.3.8 Friends and Things: A Social WoT Web Application . . . 74
2.3.9 Summary and Applications . . . 75
2.4 Composition Layer . . . 76
2.4.1 Physical Mashups in the Web of Things . . . 76
2.4.2 From Web 2.0 Mashups Editors to Physical Mashup Editors . . . 78
2.4.3 Adapting a Web 2.0 Mashup Editor to the Web of Things . . . 78
2.4.4 Requirements for Physical Mashup Editors . . . 82
2.4.5 A Platform for Physical Mashups Editors . . . 83
2.4.6 System Architecture . . . 84
2.4.7 Discussion and Summary . . . 89
2.5 Developers Perspectives on the WS-* Alternative Architecture . . . 90
2.5.1 Methodology . . . 91
2.5.2 Results . . . 93
2.6 Discussion and Summary . . . 98
2.7 Related Work . . . 99
2.7.1 Device Accessibility Layer . . . 100
2.7.2 Findability Layer . . . 101
2.7.3 Sharing Layer . . . 104
2.7.4 Composition Layer . . . 105
2.8 Summary . . . 107

In this chapter, we present the Web of Things Architecture. We describe the four layers it is based on: the Device Accessibility Layer, Findability Layer, Sharing Layer and Composition Layer as shown in Figure 2.1.

The overall goal of this architecture is to facilitate the integration of smart things with existing services on the Web and to facilitate the creation of Web applications using smart things. In particular, we formulate the following general requirements [77, 72] we would like for our architecture to fulfill:

1. It should *lower the entry barrier for developers* and foster rapid prototyping. This allows a wider range of developers, tech-savvy users (technologically skilled people) or researchers to develop on top of smart things and contributes to fostering third party (public) innovation using smart things.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0334

Bibliography

[1] K. Aberer, M. Hauswirth, and A. Salehi. Infrastructure for Data Processing in Large-Scale Interconnected Sensor Networks. In *Proc. of the International Conference on Mobile Data Management*, pages 198–205, 2007.

[2] W. Abrahamse, L. Steg, C. Vlek, and T. Rothengatter. A review of intervention studies aimed at household energy conservation. *Journal of Environmental Psychology*, 25(3):273–291, 2005.

[3] Robert Adelmann, Marc Langheinrich, and Christian Floerkemeier. A Toolkit for Bar-Code-Recognition and -Resolving on Camera Phones Jump Starting the Internet of Things. In *Proc. of the workshop on Mobile and Embedded Interactive Systems (MEIS'06) at Informatik 2006. GI Lecture Notes in Informatics Series (LNI)*, Dresden, Germany, 2006.

[4] Azucena Guillen Aguilar. *Exploring Physical Mashups on Mobile Devices*. Master thesis, ETH Zurich, Switzerland, 2010.

[5] Ryan Aipperspach, Ben Hooker, and Allison Woodruff. The heterogeneous home. *Interactions*, 16:35–38, January 2009.

[6] I. F Akyildiz, Weilian Su, Y. Sankarasubramaniam, and E. Cayirci. A survey on sensor networks. *IEEE Communications Magazine*, 40(8):102– 114, August 2002.

[7] Rosa Alarcón and Erik Wilde. RESTler: crawling RESTful services. In *Proc. of the 19th international conference on World Wide Web (WWW '10)*, pages 1051–1052, New York, NY, USA, 2010. ACM.

[8] Subbu Allamaraju. *RESTful Web Services Cookbook*. Yahoo Press, March 2010.

[9] John Allsopp. *Microformats: Empowering Your Markup for Web 2.0*. friendsofED, March 2007.

[10] Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju. *Web Services: Concepts, Architectures and Applications*. Springer, December 2010.

[11] W.T. Balke and M. Wagner. Through different eyes: assessing multiple conceptual views for querying web services. In *Proc. of the 13th international World Wide Web conference (WWW '04)*, pages 196–205, New York, NY, USA, 2004. ACM.

[12] Rajiv D. Banker, Srikant M. Datar, Chris F. Kemerer, and Dani Zweig. Software complexity and maintenance costs. *Communications of the ACM*, 36:81–94, November 1993.

[13] M. Baqer and A. Kamal. S-Sensors: Integrating physical world inputs with social networks using wireless sensor networks. In *Proc. of the 5th International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP 2009)*, pages 213–218, Melbourne, Australia, 2009.

[14] John Barton, Tim Kindberg, Hui Dai, Nissanka B Priyantha, and Fahd Al-bin ali. Sensor-enhanced mobile web clients: an XForms approach. In *Proc. of the 12th international conference on World Wide Web (WWW '03)*, pages 80–89, Budapest, Hungary, 2003. ACM.

[15] Michael Blackstock and Adrian Friday. Uniting Online Social Networks with Places and Things. In Dominique Guinard, Vlad Trifa, and Erik Wilde, editors, *Proc. of the 2nd International Workshop on the Web of Things (WoT 2011)*, San Fransisco, USA, 2011. ACM.

[16] Miodrag Bolic and David Simplot-Ryl. *RFID Systems: Research Trends and Challenges*. Wiley, September 2010.

[17] Rachel Botsman and Roo Rogers. *What's Mine Is Yours: The Rise of Collaborative Consumption*. HarperBusiness, September 2010.

[18] Mike Botts and Alex Robin. OpenGIS Sensor Model Language (SensorML) Implementation Specification, 2007.

[19] Danah M. Boyd and Nicole B. Ellison. Social Network Sites: Definition, History, and Scholarship. *Journal of ComputerMediated Communication*, 13(1):210–230, October 2008.

[20] J. Brandt, P.J. Guo, J. Lewenstein, S.R. Klemmer, and M. Dontcheva. Writing Code to Prototype, Ideate, and Discover. *IEEE Software*, 26(5):18–24, 2009.

[21] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual Web search engine. In *Proc. of the seventh international conference on World Wide Web, WWW '98*, pages 107–117, Brisbane, Australia, 1998. Elsevier Science Publishers B. V.

[22] Andreas Brodt and Daniela Nicklas. The TELAR mobile mashup platform for Nokia internet tablets. In *Proc. of the 11th international conference on Extending DataBase Technology: Advances in database technology (EDBT '08)*, pages 700–704, Nantes, France, 2008. ACM.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0336

[23] Gregor Broll, John Hamard, Massimo Paolucci, Markus Haarländer, Matthias Wagner, Sven Siorpaes, Enrico Rukzio, Albrecht Schmidt, and Kevin Wiesner. Mobile interaction with web services through associated real world objects. In *Proc. of the 9th international conference on Human computer interaction with mobile devices and services (MobileHCI '07)*, pages 319–321, New York, NY, USA, 2007. ACM.

[24] Gregor Broll, Sven Siorpaes, Enrico Rukzio, Massimo Paolucci, John Hamard, Matthias Wagner, and Albrecht Schmidt. Supporting Mobile Service Usage through Physical Mobile Interaction. In *Fifth Annual IEEE International Conference on Pervasive Computing and Communications (PerCom '07)*, pages 262–271. Ieee, 2007.

[25] J. Bungo. Embedded Systems Programming in the Cloud: A Novel Approach for Academia. *IEEE Potentials*, 30(1):17–23, 2011.

[26] T. Burbridge and M. Harrison. Security considerations in the design and peering of RFID Discovery Services. In *Proc. of the IEEE International Conference on RFID*, pages 249–256. IEEE, April 2009.

[27] Davide Carboni and Pietro Zanarini. Wireless wires: let the user build the ubiquitous computer. In *Proc. of the 6th international conference on Mobile and Ubiquitous Multimedia (MUM '07)*, pages 169–175, Oulu, Finland, 2007. ACM.

[28] Gong Chen, Nathan Yau, Mark Hansen, and Deborah Estrin. Sharing Sensor Network Data. Technical report, UC Los Angeles, 2007.

[29] Marshini Chetty, David Tran, and Rebecca E Grinter. Getting to green: understanding resource consumption in the home. In *Proc. of the 10th International Conference on Ubiquitous Computing (UbiComp '08)*, pages 242–251, Seoul, Korea, 2008. ACM.

[30] Paul Couderc. Spreading the web. In Marco Conti, Silvia Giordano, Enrico Gregori, and Stephan Olariu, editors, *Personal Wireless Communications*, volume 2775 of *LNCS*, pages 375–384. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[31] Marco Crasso, Alejandro Zunino, and Marcelo Campo. Easy web service discovery: A query-by-example approach. *Science of Computer Programming*, 71(2):144–164, April 2008.

[32] Fred D. Davis. Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology. *MIS Quarterly*, 13(3):319–340, 1989.

[33] Stephen Dawson-Haggerty, Xiaofan Jiang, and Gilman Tolle. sMAP: a simple measurement and actuation profile for physical information. In *Proc. of the 8th ACM Conference on Embedded Networked Sensor Systems (Sensys '10)*, pages 197–210, New York, NY, USA, 2010. ACM.

[34] J.D. Day and H. Zimmermann. The OSI reference model. *Proceedings of the IEEE*, 71(12):1334–1340, 1983.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0337

[35] S. de Deugd, R. Carroll, K.E. Kelly, B. Millett, and J. Ricker. SODA: Service Oriented Device Architecture. *Pervasive Computing, IEEE*, 5(3):94–96, 2006.

[36] L. de Souza, P. Spiess, D. Guinard, M. Köhler, S. Karnouskos, and D. Savio. Socrades: A web service based shop floor integration infrastructure. In Christian Floerkemeier, Marc Langheinrich, Elgar Fleisch, Friedemann Mattern, and Sanjay E. Sarma, editors, *Proc. of the Internet of Things Conference (IoT '08)*, LNCS, pages 50–67, Zurich, Switzerland, March 2008. Springer Berlin Heidelberg.

[37] Robert Dickerson, Jiakang Lu, Jian Lu, and Kamin Whitehouse. Stream Feeds - An Abstraction for the World Wide Sensor Web. In *Proc. of the Internet of Things Conference (IoT '08)*, LNCS, pages 360–375, Zurich, Switzerland, 2008. Springer Berlin Heidelberg.

[38] Bettina Dober. *Exploring Query Augmentation for the SOCRADES Application Service Catalogue*. Master thesis, University of Fribourg, Switzerland, 2009.

[39] W. Drytkiewicz, I. Radusch, S. Arbanowski, and R. Popescu-Zeletin. pREST: a REST-based protocol for pervasive systems. In *Proc. of the IEEE International Conference on Mobile Ad-hoc and Sensor Systems*, pages 340–348. IEEE, 2004.

[40] Adam Dunkels. Full TCP/IP for 8-bit architectures. In *Proc. of the 1st international conference on Mobile Systems, Applications and Services*, MobiSys '03, pages 85–98, New York, NY, USA, 2003. ACM.

[41] S. Duquennoy, G. Grimaud, and J.-J. Vandewalle. Consistency and scalability in event notification for embedded Web applications. In *Proc. of the 11th IEEE International Symposium on Web Systems Evolution (WSE '09)*, pages 89–98. IEEE, 2009.

[42] Simon Duquennoy, Gilles Grimaud, and J.J. Vandewalle. The Web of Things: interconnecting devices with high usability and performance. In *Proc. of the International Conference on Embedded Software and Systems*, pages 323–330, HangZhou, Zhejiang, China, May 2009. IEEE.

[43] Charles Engelke and Craig Fitzgerald. Replacing legacy web services with RESTful services. In *Proc. of the First International Workshop on RESTful Design (WS-REST '10)*, pages 27–30, Raleigh, North Carolina, 2010. ACM.

[44] EPCglobal. EPC Information Services (EPCIS) Version 1.0.1 Specification. Technical report, GS1, 2007.

[45] EPCglobal. The Application Level Events (ALE) Specification ver 1.1.1. Technical report, GS1, 2009.

[46] EPCglobal. EPC Tag Data Standard. Technical report, GS1, 2010.

[47] EPCglobal. Low Level Reader Protocol (LLRP) ver. 1.1. Technical report, GS1, 2010.

[48] Eran Hammer-Lahav. The OAuth 1.0 Protocol. Technical report, April 2010.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0338

[49] Thomas Erl. *Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services*. Prentice Hall, April 2004.

[50] R. Fielding. *Architectural styles and the design of network-based software architectures*. Phd thesis, 2000.

[51] R Fielding, J Gettys, J C Mogul, H Frystyk, L Masinter, P Leach, and Tim Berners-Lee. Hypertext Transfer Protocol-HTTP/1.1. Technical report, World Wide Web Consortium, 1999.

[52] Klaux Finkenzeller and D. Mueller. *RFID Handbook: Fundamentals and Applications in Contactless Smart Cards, Radio Frequency Identification and Near-Field Communication*. Wiley, second edition, 2010.

[53] C. Fischer. Feedback on household electricity consumption: a tool for saving energy? *Journal of Energy Efficiency*, 1(1):79–104, 2008.

[54] Christian Floerkemeier. EPC-Technologie vom Auto-ID Center zu EPCglobal. In Elgar Fleisch and Friedemann Mattern, editors, *Das Internet der Dinge*, pages 87–100. Springer-Verlag, 2005.

[55] Christian Floerkemeier. Integrating RFID Readers in the Enterprise IT Overview of Intra-organizational RFID System Services and Architectures. In *Integrating RFID Readers in Enterprise IT*, pages 269–295. John Wiley & Sons, Ltd, 2010.

[56] Christian Floerkemeier and Matthias Lampe. Facilitating RFID development with the accada prototyping platform. In *Fifth IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom '07)*, pages 495–500, New York, NY, USA, 2007. IEEE Computer Society.

[57] Christian Floerkemeier, Christof Roduner, and Matthias Lampe. RFID application development with the Accada middleware platform. *IEEE Systems Journal*, 1(2):82–94, December 2007.

[58] C. Frank, P. Bollinger, C. Roduner, and W. Kellerer. Objects Calling Home: Locating Objects Using Mobile Phones. In *Proc. of the International Conference on Pervasive Computing (Pervasive '07)*, Toronto, Canada, 2007.

[59] Christian Frank, Philipp Bolliger, Friedemann Mattern, and Wolfgang Kellerer. The Sensor Internet at Work: Locating Everyday Items Using Mobile Phones. *Pervasive and Mobile Computing*, 4(3):421–447, June 2008.

[60] Jon Froehlich. Promoting Energy Efficient Behaviors in the Home through Feedback : The Role of Human-Computer Interaction. *Computing Systems*, 9:10, 2009.

[61] Patrik Fuhrer and Dominique Guinard. Building a Smart Hospital using RFID Technologies. In *Proc. of the European Conference on eHealth (ECEH)*, Fribourg, Switzerland, October 2006.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0339

[62] Erich Gamma, Richard Helm, Ralph Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software*, volume 206. Addison-wesley Reading, MA, November 1995.

[63] David Gefen and Mark Keil. The impact of developer responsiveness on perceptions of usefulness and ease of use: an extension of the technology acceptance model. *SIGMIS Database*, 29:35–49, April 1998.

[64] Hans Gellersen, Carl Fischer, and Dominique Guinard. Supporting device discovery and spontaneous interaction with spatial references. *Journal of Personal and Ubiquitous Computing (PUC)*, 2009.

[65] N. Gershenfeld and D. Cohen. Internet 0: Interdevice Internetworking - End-to-End Modulation for Embedded Networks. *Circuits and Devices Magazine, IEEE*, 22(5):48–55, 2006.

[66] Antonio Goncalves. *Beginning Java EE 6 with GlassFish 3*. Apress, 2 edition, August 2010.

[67] W. I Grosky, A. Kansal, S. Nath, Jie Liu, and Feng Zhao. SenseWeb: An Infrastructure for Shared Sensing. *IEEE Multimedia*, 14(4):8–13, December 2007.

[68] Dominique Guinard. Mashing up your web-enabled home. In *Proc. of the 10th International conference on Current trends in Web Engineering (ICWE '10)*, pages 442–446, Vienna, Austria, July 2010. Springer-Verlag.

[69] Dominique Guinard, Oliver Baecker, and Florian Michahelles. Supporting a mobile lost and found community. In *Proc. of the 10th international conference on Human Computer Interaction with mobile devices and services (Mobile HCI '08)*, pages 407–410. ACM, September 2008.

[70] Dominique Guinard, Oliver Baecker, Patrik Spiess, Stamatis Karnouskos, Moritz Koehler, Luciana Moreira Sa De Souza, Domnic Savio, and Mihai Vlad Trifa. On-demand provisionning of services running on embedded devices, Patent, 2010.

[71] Dominique Guinard, Mathias Fischer, and Vlad Trifa. Sharing using social networks in a composable web of things. In *Proc. of the First IEEE International Workshop on the Web of Things (WoT 2010)*, pages 702–707, Mannheim, Germany, March 2010. IEEE.

[72] Dominique Guinard, Christian Floerkemeier, and Sanjay Sarma. Cloud Computing, REST and Mashups to Simplify RFID Application Development and Deployment. In *Proc. of the 2nd International Workshop on the Web of Things (WoT 2011)*, San Fransisco, USA, June 2011. ACM.

[73] Dominique Guinard, Iulia Ion, and Simon Mayer. In Search of an Internet of Things Service Architecture: REST or WS-*? A Developers' Perspective. In *Proc. of Mobiquitous 2011 (8th International ICST Conference on Mobile and Ubiquitous Systems).*, Copenhagen, Denmark, 2011.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0340

[74] Dominique Guinard, Mathias Mueller, and Jacques Pasquier. Giving RFID a REST: Building a Web-Enabled EPCIS. In *Proc. of the International conference on the Internet of Things (IoT '10)*, LNCS, Tokyo, Japan, November 2010. Springer Berlin / Heidelberg.

[75] Dominique Guinard, Mathias Mueller, and Vlad Trifa. *RESTifying Real-World Systems: a Practical Case Study in RFID*, chapter 16. Springer, August 2011.

[76] Dominique Guinard and Vlad Trifa. Towards the web of things: Web mashups for embedded devices. In *Proc. of the Second Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web (MEM '09)*, WWW '09, Madrid, Spain, April 2009. ACM.

[77] Dominique Guinard, Vlad Trifa, Stamatis Karnouskos, Patrik Spiess, and Domnic Savio. Interacting with the SOA-Based Internet of Things: Discovery, Query, Selection, and On-Demand Provisioning of Web Services. *IEEE Transactions on Services Computing*, 3(3):223–235, February 2010.

[78] Dominique Guinard, Vlad Trifa, Friedemann Mattern, and Erik Wilde. From the Internet of Things to the Web of Things: Resource-oriented Architecture and Best Practices. In Dieter Uckelmann, Mark Harrison, and Florian Michahelles, editors, *Architecting the Internet of Things*, pages 97–129. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[79] Dominique Guinard, Vlad Trifa, Thomas Pham, and Olivier Liechti. Towards physical mashups in the web of things. In *Proc. of the 6th International Conference on Networked Sensing Systems (INSS '09)*, pages 1–4, Pittsburgh, USA, June 2009. IEEE.

[80] Dominique Guinard, Vlad Trifa, Patrik Spiess, Bettina Dober, and Stamatis Karnouskos. Discovery and On-Demand Provisionning of Real-World Web Services. In *Proc. of the IEEE International Conference on Web Services (ICWS '09)*, Los Angeles, California, USA, July 2009.

[81] Dominique Guinard, Vlad Trifa, and Erik Wilde. A Resource Oriented Architecture for the Web of Things. In *Proc. of the 2nd International Conference on the Internet of Things (IoT 2010)*, LNCS, Tokyo, Japan, November 2010. Springer Berlin / Heidelberg.

[82] Dominique Guinard, Felix von Reischach, and Florian Michahelles. MobileIoT Toolkit: Connecting the EPC Network to MobilePhones. In *Proceedings of Mobile Interaction with the Real World (MIRW '08)*, Amsterdam, Netherlands, September 2008. The University of Oldenburg.

[83] Dominique Guinard, Markus Weiss, and Vlad Trifa. Are you energy-efficient? sense it on the web. In *Adjunct Proc. of the International Conference on Pervasive Computing (Pervasive '09)*, Nara, Japan, May 2009. Springer.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0341

[84] V. Gupta, P. Udupi, and A. Poursohi. Early lessons from building Sensor.Network: an open data exchange for the web of things. In *Proc. of the First International Workshop on the Web of Things (WoT 2010)*, pages 738–744. IEEE, 2010.

[85] Vipul Gupta, Ron Goldman, and Poornaprajna Udupi. A network architecture for the Web of Things. In *Proc. of the Second International Workshop on Web of Things (WoT 2011)*, WoT '11, pages 3:1–3:6, San Francisco, California, 2011. ACM.

[86] Marc J Hadley. Web application description language (WADL). Technical report, Sun Microsystems, Inc., 2006.

[87] Richard Hall, Karl Pauls, Stuart McCulloch, and David Savage. *OSGi in Action: Creating Modular Applications in Java*. Manning Publications, April 2011.

[88] Mark Halvorson and Andy Smith. OpenSocial 2.0 Specification. Technical report, Open Social Working Group, 2010.

[89] Haodong Wang, C. C Tan, and Qun Li. Snoogle: A Search Engine for Pervasive Environments. *IEEE Transactions on Parallel and Distributed Systems*, 21(8):1188–1202, August 2010.

[90] B. Hartmann and S. Doorley. Hacking, mashing, gluing: understanding opportunistic design. *IEEE Pervasive Computing*, 7(3):46–54, 2008.

[91] S. Helal, W. Mann, H. El-Zabadani, J. King, Y. Kaddoura, and E. Jansen. The Gator Tech Smart House: a programmable pervasive space. *Computer*, 38(3):50–60, 2005.

[92] Johannes Helander. Deeply embedded XML communication: towards an interoperable and seamless world. In *Proc. of the 5th ACM international conference on embedded software (EMSOFT '05)*, pages 62–67, Jersey City, NJ, USA, 2005. ACM.

[93] M. Hepp, K. Siorpaes, and D. Bachlechner. Harvesting Wiki Consensus: Using Wikipedia Entries as Vocabulary for Knowledge Management. *IEEE Internet Computing,*, 11(5):54–65, 2007.

[94] Soojung Hong. *Mobile Discovery in a Web of Things*. Master thesis, ETH Zurich, Switzerland, 2010.

[95] A. Hornsby and E. Bail. μXMPP: Lightweight implementation for low power operating system Contiki. In *International Conference on Ultra Modern Telecommunications (ICUMT '09)*, pages 1–5. IEEE, October 2009.

[96] A. Hornsby and R. Walsh. From instant messaging to cloud computing, an XMPP review. pages 1–6. IEEE, June 2010.

[97] Volker Hoyer, Katarina Stanoesvka-Slabeva, Till Janner, and Christoph Schroth. Enterprise Mashups: Design Principles towards the Long Tail of User Needs. In *IEEE International Conference on Services Computing (SCC '08)*, volume 2, pages 601–602, 2008.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0342

[98] J.W. Hui and D.E. Culler. Extending IP to low-power, wireless personal area networks. *IEEE Internet Computing*, 12(4):37–45, 2008.

[99] J.W. Hui and D.E. Culler. IP is dead, long live IP for wireless sensor networks. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 15–28, Raleigh, NC, USA, 2008. ACM.

[100] Jonathan J Hull, Xu Liu, Berna Erol, Jamey Graham, and Jorge Moraleda. Mobile image recognition: architectures and tradeoffs. In *Proc. of the Eleventh Workshop on Mobile Computing Systems & Applications (HotMobile '10)*, pages 84–88, New York, NY, USA, 2010. ACM.

[101] Iulia Ion, Marc Langheinrich, Ponnurangam Kumaraguru, and Srdjan Čapkun. Influence of user perception, security needs, and social factors on device pairing method choices. In *Proc. of SOUPS 2010*, SOUPS '10, pages 6:1–6:13, Redmond, Washington, USA, 2010. ACM.

[102] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart. HTTP Authentication: Basic and Digest Access Authentication. Technical report, IETF, 1999.

[103] F. Jammes and H. Smit. Service-oriented paradigms in industrial automation. *IEEE Transactions on Industrial Informatics*, 1(1):62–70, 2005.

[104] François Jammes, Antoine Mensch, and Harm Smit. Service-oriented device communications using the devices profile for web services. In *Proceedings of the 3rd international workshop on Middleware for pervasive and ad-hoc computing*, MPAC '05, pages 1–8, New York, NY, USA, 2005. ACM.

[105] Xiaofan Jiang, Stephen Dawson-Haggerty, Prabal Dutta, and David Culler. Design and implementation of a high-fidelity AC metering network. In *Proc. of the 2009 International Conference on Information Processing in Sensor Networks (IPSN '09)*, pages 253–264. IEEE Computer Society, 2009.

[106] A. Juels. RFID security and privacy: a research survey. *IEEE Journal on Selected Areas in Communications*, 24(2):381–394, February 2006.

[107] Andreas Kamilaris, Nicolas Iannarilli, Vlad Trifa, and Andreas Pitsillides. Bridging the Mobile Web and the Web of Things in Urban Environments. In *Proc. of the first Workshop on the Urban Internet of Things (UrbanIoT '10)*, Tokyo, Japan, 2010.

[108] S Karnouskos, D Savio, P Spiess, D Guinard, V Trifa, and O Baecker. Real-world Service Interaction with Enterprise Systems in Dynamic Manufacturing Environments. In Lyes Benyoucef and Bernard Grabot, editors, *Artificial Intelligence Techniques for Networked Manufacturing Enterprises Management*, Springer Series in Advanced Manufacturing, pages 423–457. Springer London, 2010.

[109] A. Katasonov and M. Palviainen. Towards ontology-driven development of applications for smart environments. In *Proc. the International Conference on Pervasive*

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0343

Computing and Communications Workshops (PERCOM Workshops 2010), pages 696–701. IEEE, April 2010.

[110] Tim Kindberg, John Barton, Jeff Morgan, Gene Becker, Debbie Caswell, Philippe Debaty, Gita Gopal, Marcos Frid, Venky Krishnan, Howard Morris, and Others. People, places, things: Web presence for the real world. *Mobile Networks and Applications*, 7(5):365–376, 2002.

[111] Jacek Kopecký, Karthik Gomadam, and Tomas Vitvar. hRESTS: An HTML Microformat for Describing RESTful Web Services. pages 619–625. IEEE Computer Society, 2008.

[112] Tiiu Koskela, Kaisa Väänänen-Vainio-Mattila, and Lauri Lehti. Home Is Where Your Phone Is: Usability Evaluation of Mobile Phone UI for a Smart Home. In Stephen Brewster and Mark Dunlop, editors, *Mobile Human-Computer Interaction MobileHCI 2004*, volume 3160 of *LNCS*, pages 74–85. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[113] Mathias Kovatsch, Markus Weiss, and Dominique Guinard. Embedding internet technology for home automation. In *Emerging Technologies and Factory Automation (ETFA), 2010 IEEE Conference on*, pages 1–8, Bilbao, Spain, September 2010. IEEE.

[114] D. Kristol and L. Montulli. HTTP State Management Mechanism. Technical report, 1997.

[115] Kirk L Kroeker. The evolution of virtualization. *Communications of the ACM*, 52:18–20, March 2009.

[116] Roland Kübert, Gregory Katsaros, and Tinghe Wang. A RESTful implementation of the WS-agreement specification. In *Proc. of the Second International Workshop on RESTful Design (WS-REST '11)*, WS-REST '11, pages 67–72, New York, NY, USA, 2011. ACM.

[117] Chris Kürschner, Cosmin Condea, Oliver Kasten, and Frédéric Thiesse. Discovery Service Design in the EPCglobal Network. In Christian Floerkemeier, Marc Langheinrich, Elgar Fleisch, Friedemann Mattern, and Sanjay E. Sarma, editors, *Proc. of the Internet of Things Conference (IoT '08)*, volume 4952 of *LNCS*, pages 19–34, Zurich, Switzerland, 2008. Springer Berlin Heidelberg.

[118] Marc Langheinrich. *Personal Privacy in Ubiquitous Computing – Tools and System Support*. PhD thesis, ETH Zurich, Zurich, Switzerland, May 2005.

[119] Jon Lathem, Karthik Gomadam, and Amit P Sheth. SA-REST and (S)mashups: Adding Semantics to RESTful Services. In *Proc. of the International Conference on Semantic Computing (ICSC '07)*, pages 469–476. IEEE Computer Society, 2007.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0344

[120] J.S. Lee, Y.W. Su, and C.C. Shen. A comparative study of wireless protocols: Bluetooth, UWB, ZigBee, and Wi-Fi. In *Proc. of the 33rd Annual Conference of the IEEEIndustrial Electronics Society (IECON 2007)*, pages 46–51. IEEE, 2007.

[121] Mark Lentczner and Donovan Preston. Reverse HTTP. Technical report, IETF, March 2009.

[122] Joshua Lifton, Mark Feldmeier, Yasuhiro Ono, Cameron Lewis, and Joseph A Paradiso. A platform for ubiquitous sensor deployment in occupational and domestic environments. In *Proc. of the 6th international conference on Information processing in sensor networks (IPSN '07)*, pages 119–127, Cambridge, Massachusetts, USA, 2007. ACM.

[123] Tao Lin, Hai Zhao, Jiyong Wang, Guangjie Han, and Jindong Wang. An Embedded Web Server for Equipments. In *Proc. of the International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN'04)*, Hong Kong, Hong Kong, 2004.

[124] Alexander Linden, Jackie Fenn, David W McCoy, David W Cearley, Nikos Drakos, Jim Tully, Monica Basso, Phillip Redman, Erik Dorr, Allen Weiner, Kenshi Tazaki, Ant Allan, Rafe John Graham Ball, Jim Sinur, Michael A Silver, Martin Gilliland, Carl Claunch, Ray Valdes, Betsy Burton, Jeff Woods, Nick Jones, Jeffrey Mann, Whit Andrews, John Pescatore, Christophe Uzureau, Mary Knox, Rita E Knox, Van L Baker, Mike McGuire, Leslie Fiering, Christopher Ambrose, Steve Cramoysan, Bern Elliot, Bob Hafner, Yefim V Natis, Benoit J Lheureux, Howard J Dresner, Michael J Blechar, Brian Gammage, and Mark A Margevicius. Hype Cycle for Emerging Technologies 2005, August 2005.

[125] Peter Lubbers, Brian Albers, and Frank Salim. *Pro HTML 5 Programming*. Apress, March 2010.

[126] T. Luckenbach, P. Gober, S. Arbanowski, A. Kotsopoulos, and K. Kim. TinyREST: A protocol for integrating sensor networks into the internet. In *Proc. of REALWSN*, Stockholm, Sweden, 2005. Citeseer.

[127] Liqian Luo, Aman Kansal, Suman Nath, and Feng Zhao. Sharing and exploring sensor streams over geocentric interfaces. In *Proc. of the 16th ACM SIGSPATIAL international conference on advances in geographic information systems (GIS '08)*, pages 1–10, Irvine, California, 2008. ACM.

[128] Urs Maeder. *Smart Lost & Found Services for Insurers*. Master thesis, University of St-Gallen, Switzerland, 2007.

[129] M. Marin-Perianu, N. Meratnia, P. Havinga, L.M.S. de Souza, J. Muller, P. Spieß, S. Haller, T. Riedel, C. Decker, and G. Stromberg. Decentralized enterprise systems: a multiplatform wireless sensor network approach. *IEEE Wireless Communications*, 14(6):57–66, 2007.

[130] Toshiyuki Masui, Koji Tsukada, and Itiro Siio. MouseField: A Simple and Versatile Input Device for Ubiquitous Computing. In Nigel Davies, Elizabeth D. Mynatt, and

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0345

Itiro Siio, editors, *Proc. of the Conference on Ubiquitous Computing (Ubicomp '04)*, volume 3205 of *LNCS*, pages 319–328, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[131] Mathias Fischer. *Social Access Controller: Sharing, Protecting and Publishing RESTful Resources through Social Networks*. Semester thesis, ETH Zurich, Zurich, Switzerland, July 2009.

[132] Kieran Mathieson, Eileen Peacock, and Wynne W Chin. Extending the technology acceptance model: the influence of perceived user resources. *ACM SIGMIS Database*, 32:86–112, July 2001.

[133] Friedemann Mattern. Die technische Basis für das Internet der Dinge. In Elgar Fleisch and Friedemann Mattern, editors, *Das Internet der Dinge*, pages 39–66. Springer-Verlag, 2005.

[134] Friedemann Mattern and Christian Floerkemeier. From the Internet of Computers to the Internet of Things. In Kai Sachs, Ilia Petrov, and Pablo Guerrero, editors, *Active Data Management to Event-Based Systems and More*, volume 6462 of *LNCS*, pages 242–259. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2010.

[135] Friedemann Mattern, Thorsten Staake, and Markus Weiss. ICT for green: how computers can help us to conserve energy. In *Proc. of the 1st International Conference on Energy-Efficient Computing and Networking (e-Energy '10)*, e-Energy '10, pages 1–10, Passau, Germany, 2010. ACM.

[136] E. Michael Maximilien. Mobile Mashups: Thoughts, Directions, and Challenges. In *Proc. of the IEEE International Conference on Semantic Computing*, pages 597–600, 2008.

[137] Simon Mayer. *Deployment Support for an Infrastructure for Web-enabled Devices*. Master thesis, ETH Zurich, Switzerland, 2010.

[138] Simon Mayer and Dominique Guinard. An Extensible Discovery Service for Smart Things. In *Proc. of the 2nd International Workshop on the Web of Things (WoT 2011).*, San Fransisco, USA, June 2011. ACM.

[139] Michelle L Mazurek, J. P Arsenault, Joanna Bresee, Nitin Gupta, Iulia Ion, Christina Johns, Daniel Lee, Yuan Liang, Jenny Olsen, Brandon Salmon, Richard Shay, Kami Vaniea, Lujo Bauer, Lorrie Faith Cranor, Gregory R Ganger, and Michael K Reiter. Access Control for Home Data Sharing: Attitudes, Needs and Practices. In *Proc. of the 28th international conference on Human factors in computing systems (CHI 2010)*, CHI '10, pages 645–654, New York, NY, USA, 2010. ACM.

[140] T. Mikkonen and A. Salminen. Towards Pervasive Mashups in Embedded Devices. In *Proc. of the 16th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'10)*, pages 35–42, 2010.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0346

[141] Andrew D Miller and W. Keith Edwards. Give and take: a study of consumer photo-sharing culture and practice. In *Proc. of the SIGCHI conference on Human factors in computing systems*, CHI '07, pages 347–356, New York, NY, USA, 2007. ACM.

[142] Richard Monson-Haefel. *J2EE Web Services: XML SOAP WSDL UDDI WS-I JAX-RPC JAXR SAAJ JAXP*. Addison-Wesley Professional, October 2003.

[143] Richard Monson-Haefel and Bill Burke. *Enterprise JavaBeans 3.0*. O'Reilly Media, 5 edition, May 2006.

[144] Peter Morville. *Ambient Findability: What We Find Changes Who We Become*. O'Reilly Media, October 2005.

[145] Peter Morville and Louis Rosenfeld. *Information Architecture for the World Wide Web: Designing Large-Scale Web Sites*. O'Reilly Media, third edit edition, December 2006.

[146] Luca Mottola and Gian Pietro Picco. Programming wireless sensor networks: Fundamental concepts and state of the art. *ACM Comput. Surv.*, 43(3):19:1–19:51, April 2011.

[147] Mathias Müller. *Design and Implementation of a Web-enabled Electronic Product*. Master thesis, University of Fribourg, 2009.

[148] Geoff Mulligan. The 6LoWPAN architecture. In *Proc. of the 4th workshop on Embedded networked sensors (EmNets '07)*, EmNets '07, pages 78–82, Cork, Ireland, 2007. ACM.

[149] Lukas Naef. *ClickScript a visual programming language in the browser*. Master thesis, ETH Zurich, Switzerland, 2009.

[150] L. O'Gorman and T. Pavlidis. Auto ID technology: From barcodes to biometrics. *IEEE Robotics & Automation Magazine*, 6(1):4–6, 1999.

[151] Benedikt Ostermaier, Kay Römer, Friedemann Mattern, Michael Fahrmair, and Wolfgang Kellerer. A Real-Time Search Engine for the Web of Things. In *Proceedings of Internet of Things 2010 International Conference (IoT 2010)*, Tokyo, Japan, November 2010.

[152] Danny Parker, David Hoak, Jamie Cummings, and Florida Solar. Pilot Evaluation of Energy Savings and Persistence from Residential Energy Demand Feedback Devices in a Hot Climate. In *Proc of the ACEEE Summer Study on Energy Efficiency in Buildings (ACEEE 2010)*, pages 245–259, 2010.

[153] Cesare Pautasso and Erik Wilde. Why is the web loosely coupled?: a multi-faceted metric for service design. In *Proc. of the 18th international conference on World Wide Web (WWW '09)*, pages 911–920, Madrid, Spain, April 2009. ACM.

[154] Cesare Pautasso, Olaf Zimmermann, and Frank Leymann. Restful web services vs. big web services: making the right architectural decision. In *Proc. of the 17th*

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0347

international conference on World Wide Web (WWW '08), pages 805–814, New York, NY, USA, 2008. ACM.

[155] Thomas Pham. *Resource Oriented Architecture in Wireless Sensor Network*. Bachelor thesis, University of Applied Science of Western Switzerland, 2008.

[156] Antonio Pintus, Davide Carboni, and Andrea Piras. The Anatomy of a Large Scale Social Web for Internet Enabled Objects. In Dominique Guinard, Vlad Trifa, and Erik Wilde, editors, *Proc. of the 2nd International Workshop on the Web of Things (WoT 2011)*, San Fransisco, USA, 2011. ACM.

[157] N.B. Priyantha, Aman Kansal, Michel Goraczko, and Feng Zhao. Tiny web services: design and implementation of interoperable and evolvable sensor networks. In *Proc. of the 6th ACM conference on Embedded Network Sensor Systems (SenSys '08)*, pages 253–266, Raleigh, NC, USA, 2008. ACM.

[158] T Quack, H Bay, and L Van Gool. Object Recognition for the Internet of Things. In Christian Floerkemeier, Marc Langheinrich, Elgar Fleisch, Friedemann Mattern, and Sanjay E Sarma, editors, *First International Conference on the Internet of Things (IoT 2008)*, volume 4952 of *Lecture Notes in Computer Science*, pages 230–246. Springer-Verlag New York Inc, 2008.

[159] Jun Rekimoto, Brygg Ullmer, and Haruo Oba. DataTiles: a modular platform for mixed physical and graphical interactions. In *Proc. of the SIGCHI conference on Human factors in computing systems (CHI '01)*, CHI '01, pages 269–276, Seattle, Washington, United States, 2001. ACM.

[160] Leonard Richardson and Sam Ruby. *RESTful web services*. O'Reilly Media, May 2007.

[161] T. Riedel, N. Fantana, A. Genaid, D. Yordanov, H.R. Schmidtke, and M. Beigl. Using web service gateways and code generation for sustainable IoT system development. In *Internet of Things (IOT), 2010*, pages 1–8. IEEE, 2010.

[162] Joel J. P. C. Rodrigues and Paulo A. C. S. Neves. A survey on IP-based wireless sensor network solutions. *International Journal of Communication Systems*, 2010.

[163] Marc Roelands, Laurence Claeys, Marc Godon, Marjan Geerts, Mohamed Ali Feki, and Lieven Trappeniers. Enabling the Masses to Become Creative in Smart Spaces. In Dieter Uckelmann, Mark Harrison, and Florian Michahelles, editors, *Architecting the Internet of Things*, pages 37–64. Springer Berlin Heidelberg, 2011.

[164] Michael Rohs. *Linking Physical and Virtual Worlds with Visual Markers and Handheld Devices*. Phd thesis, ETH Zurich, Zurich, Switzerland, August 2005.

[165] Kay Römer, Thomas Schoch, Friedemann Mattern, and Thomas Dübendorfer. Smart Identification Frameworks for Ubiquitous Computing Applications. *Wireless Networks*, 10(6):689–700, December 2004.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0348

[166] K. Romer, B. Ostermaier, F. Mattern, M. Fahrmair, and W. Kellerer. Real-Time Search for Real-World Entities: A Survey. *Proceedings of the IEEE*, 98(11):1887–1902, 2010.

[167] Enrico Rukzio, Gregor Broll, Karin Leichtenstern, and Albrecht Schmidt. *Mobile Interaction with the Real World: An Evaluation and Comparison of Physical Mobile Interaction Techniques*, pages 1–18. Springer Berlin / Heidelberg, 2007.

[168] Daniel Salber, A.K. Dey, and G.D. Abowd. The context toolkit: aiding the development of context-enabled applications. In *Proc. of the SIGCHI conference on Human factors in computing systems: the CHI is the limit*, pages 434–441, Pittsburgh, Pennsylvania, United States, 1999. ACM.

[169] S. Sarma, D. Brock, and D. Engels. Radio frequency identification and the electronic product code. *IEEE Micro*, 21(6):50–54, 2001.

[170] Sanjay Sarma, David L Brock, and Kevin Ashton. The Networked Physical World. Technical report, MIT Auto-ID Labs, 2001.

[171] Albrecht Schmidt, Michael Beigl, and H.W. Gellersen. There is more to context than location. *Computers & Graphics*, 23(6):893–901, November 1999.

[172] Patrick Schmitt. *Adoption und Diffusion neuer Technologien am Beispiel der Radiofrequenz-Identifikation (RFID)*. Phd thesis, ETH Zurich, 2008.

[173] Lars Schor, Philipp Sommer, and Roger Wattenhofer. Towards a zero-configuration wireless sensor network architecture for smart buildings. In *Proc. of the First ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings (BuildSys '09)*, pages 31–36. ACM, November 2009.

[174] P. Schramm, E. Naroska, P. Resch, J. Platte, H. Linde, G. Stromberg, and T. Sturm. A service gateway for networked sensor systems. *IEEE Pervasive Computing*, 3(1):66– 74, March 2004.

[175] S. Senecal. The influence of online product recommendations on consumers' online choices. *Journal of Retailing*, 80(2):159, 2004.

[176] Z. Shelby. Embedded web services. *IEEE Wireless Communications*, 17(6):52–57, December 2010.

[177] A. P Sheth, K. Gomadam, and J. Lathem. SA-REST: Semantically Interoperable and Easier-to-Use Services and Mashups. *IEEE Internet Computing*, 11(6):91–94, December 2007.

[178] A.S. Shirazi, C. Winkler, and A. Schmidt. SENSE-SATION: An extensible platform for integration of phones into the Web. In *Proc. of the Internet of Things 2010 International Conference (IoT 2010)*, pages 1–8, 2010.

[179] Shwetak Patel, Thomas Robertson, Julie Kientz, Matthew Reynolds, and Gregory Abowd. At the Flick of a Switch: Detecting and Classifying Unique Electrical

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0349

Events on the Residential Power Line. In *Proc. of the International Conference on Ubiquitous Computing (UbiComp 2007)*, pages 271–288, 2007.

[180] Meredith M Skeels and Jonathan Grudin. When social networks cross boundaries: a case study of workplace use of facebook and linkedin. In *Proc. of the ACM International Conference on Supporting Group Work (Group '09)*, pages 95–104, Sanibel Island, Florida, USA, 2009. ACM.

[181] H. Song, Doreen Cheng, A. Messer, and S. Kalasapur. Web Service Discovery Using General-Purpose Search Engines. In *Proc. of the IEEE International Conference on Web Services (ICWS 2007)*, pages 265–271, 2007.

[182] Steve Souders. *High Performance Web Sites: Essential Knowledge for Front-End Engineers*. O'Reilly Media, Inc., September 2007.

[183] Patrik Spiess, Stamatis Karnouskos, Dominique Guinard, Domnic Savio, Oliver Baecker, Luciana Moreira Sa de Souza, and Vlad Trifa. SOA-based Integration of the Internet of Things in Enterprise Services. In *Proc of the IEEE International Conference on Web Services (ICWS 2009)*, Los Angeles, California, USA, July 2009.

[184] Katarina Stanoevska-Slabeva and Thomas Wozniak. Cloud Basics An Introduction to Cloud Computing. In Katarina Stanoevska-Slabeva, Thomas Wozniak, and Santi Ristol, editors, *Grid and Cloud Computing*, chapter 4, pages 47–61. Springer, Berlin, Heidelberg, 2010.

[185] V. Stirbu. Towards a RESTful Plug and Play Experience in the Web of Things. In *Proc. of the IEEE International Conference on Semantic Computing (ICSC '08)*, pages 512–517, 2008.

[186] J. Swartz. The growing magic of automatic identification. *IEEE Robotics & Automation Magazine*, 6(1):20–23, 56, 1999.

[187] C Tan, Bo Sheng, and Haodong Wang. Microsearch: When search engines meet small devices. In *Pervasive Computing*, Sydney, Australia, 2008. Springer-Verlag.

[188] Schmid Thomas, Young Cho, and Mani B. Srivastava. Exploiting Social Networks for Sensor Data Sharing with SenseShare. Technical report, CENS 5th Annual Research Review, 2007.

[189] Ken Traub, Felice Armenio, Henri Barthel, Paul Dietrich, John Duker, Christian Floerkemeier, John Garrett, Mark Harrison, Bernie Hogan, Jin Mitsugi, Josef Preishuber-Pfluegl, Oleg Ryaboy, Sanjay Sarma, KK Suen, and John Williams. The EPCglobal Architecture Framework. Technical report, EPCglobal, 2010.

[190] Vlad Trifa, Dominique Guinard, Philipp Bolliger, and Samuel Wieland. Design of a Web-based distributed location-aware infrastructure for mobile devices. In *Proc. of the First International Workshop on the Web of Things (WoT 2010)*, pages 714–719, Mannheim, Germany, March 2010. IEEE.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0350

[191] Vlad Trifa, Dominique Guinard, Vlatko Davidovski, Andreas Kamilaris, and Ivan Delchev. Web-based Messaging Mechanisms for Open and Scalable Distributed Sensing Applications. In *Proceedings the International Conference on Web Engineering (ICWE 2010)*, Vienna, Austria, July 2010.

[192] Vlad Trifa, Dominique Guinard, and Simon Mayer. *Leveraging the Web to Build a Distributed Location-aware Infrastructure for the Real World*, chapter 17. Springer, August 2011.

[193] Vlad Trifa, Samuel Wieland, Dominique Guinard, and T.M. Bohnert. Design and implementation of a gateway for web-based interaction and management of embedded devices. In *Proceedings of the 2nd International Workshop on Sensor Network Engineering (IWSNE 09)*, Marina del Rey, CA, USA, June 2009.

[194] Brygg Ullmer, Hiroshi Ishii, and Dylan Glas. mediaBlocks: physical containers, transports, and controls for online media. In *Proc. of the 25th annual conference on Computer graphics and interactive techniques (SIGGRAPH '98)*, pages 379–386, New York, NY, USA, 1998. ACM.

[195] M M Van Raaij Theo and W Fred. A behavioral model of residential energy use. *Journal of Economic Psychology*, 3(1):39–63, 1983.

[196] J.P. Vasseur and Adam Dunkels. *Interconnecting smart objects with IP: the next internet*. Morgan Kaufmann Publishers Inc., June 2010.

[197] Juan Vazquez and Diego Lopez-de Ipina. Social Devices: Autonomous Artifacts That Communicate on the Internet. In *Proc. of the 1st international conference on the Internet of Things (IoT 2008)*, pages 308–324, 2008.

[198] Jo Vermeulen, Kris Luyten, Karin Coninx, and Ruben Thys. Tangible Mashups: Exploiting Links between the Physical and Virtual World. In *Proc. of the 1st International Workshop on System Support for the Internet of Things (WoSSIoT'07)*, 2007.

[199] Jo Vermeulen, Ruben Thys, Kris Luyten, and Karin Coninx. Making Bits and Atoms Talk Today. In *Proc. of the 1st International Workshop on Design and Integration Principles for Smart Objects (DI PSO 2007)*, 2007.

[200] Felix von Reischach, Dominique Guinard, Florian Michahelles, and Elgar Fleisch. A mobile product recommendation system interacting with tagged products. In *Proc. of PerCom 2009 (IEEE International Conference on Pervasive Computing and Communications)*, Galveston, Texas, USA, March 2009.

[201] Felix Von Reischach, Florian Michahelles, Dominique Guinard, Robert Adelmann, Elgar Fleisch, and A. Schmidt. An evaluation of product identification techniques for mobile phones. In *Proc. of the Conference in Human-Computer Interaction (INTERACT 2009)*, pages 804–816, Uppsala, August 2009. Springer.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0351

[202] Roy Want, Kenneth P Fishkin, Anuj Gujar, and Beverly L Harrison. Bridging physical and virtual worlds with electronic tags. In *Proc. of the SIGCHI conference on Human factors in computing systems (CHI '99)*, pages 370–377, Pittsburgh, Pennsylvania, United States, 1999. ACM.

[203] Mark Weiser. The computer for the 21st century. *Scientific American*, 3:94–104, July 1991.

[204] Markus Weiss and Dominique Guinard. Increasing energy awareness through web-enabled power outlets. In *Proceedings of the 9th International Conference on Mobile and Ubiquitous Multimedia*, page 20, Limassol, Cyprus, December 2010. ACM.

[205] Markus Weiss, Dominique Guinard, T. Staake, and Wolf Roediger. eMeter: An interactive energy monitor. In *Adjunct Proc. of the International Conference on Ubiquitous Computing (UbiComp 2009)*, Orlando, Florida, USA, September 2009.

[206] Markus Weiss, Friedemann Mattern, Tobias Graml, Thorsten Staake, and Elgar Fleisch. Handy feedback: connecting smart meters with mobile phones. In *Proc. of the 8th International Conference on Mobile and Ubiquitous Multimedia (MUM '09)*, MUM '09, pages 15:1–15:4, Cambridge, United Kingdom, 2009. ACM.

[207] E. Welbourne, L. Battle, G. Cole, K. Gould, K. Rector, S. Raymer, M. Balazinska, and G. Borriello. Building the Internet of Things Using RFID: The RFID Ecosystem Experience. *IEEE Internet Computing*, 13(3):48–55, June 2009.

[208] T. Wiechert, F. Thiesse, F. Michahelles, P. Schmitt, and E. Fleisch. Connecting mobile phones to the Internet of things: A discussion of compatibility issues between EPC and NFC. In *Americas Conference on Information Systems, AMCIS*, Keystone, Colorado, USA, 2007.

[209] Erik Wilde. Putting Things to REST. Technical report, School of Information, UC Berkeley, November 2007.

[210] Erik Wilde. Feeds as Query Result Serializations. Technical Report 2009-030, April 2009.

[211] Jakkhupan Worapot, Yuefeng Li, and Arch-Int Somjit. Design and implementation of the EPC Discovery Services with confidentiality for multiple data owners. In *Proc of the IEEE International Conference on RFID-Technology and Applications (RFID-TA 2010)*, pages 19–25. IEEE, June 2010.

[212] Ke Xu, Xiaoqi Zhang, Meina Song, and Junde Song. Mobile Mashup: Architecture, Challenges and Suggestions. In *Proc. of the International Conference on Management and Service Science (MASS '09)*, pages 1–4, Beijing, China, September 2009.

[213] Kok-Kiong Yap, Vikram Srinivasan, and Mehul Motani. MAX: human-centric search of the physical world. In *Proc. of the 3rd international conference on embedded networked sensor systems (SenSys '05)*, pages 166–179, San Diego, California, USA, 2005. ACM.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0352

[214] Dogan Yazar and Adam Dunkels. Efficient application integration in IP-based sensor networks. In *Proceedings of the First ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*, page 4348, Berkeley, CA, USA, November 2009.

[215] Raymond Yee. *Pro Web 2.0 Mashups: Remixing Data and Web Services*. Apress, March 2008.

[216] Jin Yu, Boualem Benatallah, Fabio Casati, and Florian Daniel. Understanding Mashup Development. *IEEE Internet Computing*, 12(5):44–52, 2008.

[217] Shuai Zhang, Shufen Zhang, Xuebin Chen, and Xiuzhen Huo. Cloud Computing Research and Development Trend. In *Second International Conference on Future Networks (ICFN '10)*, pages 93–97. IEEE, January 2010.

[218] Justin Zobel, Alistair Moffat, and Kotagiri Ramamohanarao. Inverted files versus signature files for text indexing. *ACM Transactions on Database Systems (TODS)*, 23:453–490, December 1998.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0353

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0354

Referenced Web Ressources

[219] Amazon web services. http://aws.amazon.com/.

[220] American energy review (EIA) for 2010. http://www.eia.gov/totalenergy/data/annual/.

[221] Apache abdera atompub server. http://abdera.apache.org/.

[222] Apache tomcat. http://tomcat.apache.org/.

[223] Atmosphere: Web push abstraction framework. http://atmosphere.java.net/.

[224] autowot - a toolkit for the rapid integration of smart devices into the web of things - google project hosting. http://code.google.com/p/autowot/.

[225] The contiki operating system - instant contiki. http://www.sics.se/contiki/instant-contiki.html.

[226] Craigslist. http://www.craigslist.com.

[227] Dpws specification 1.1. http://docs.oasis-open.org/ws-dd/dpws/1.1/os/wsdd-dpws-1.1-spec-os.pdf.

[228] EnergieVisible: our consumption is visible! opensource software for sensing and visualizing electricity consumption. http://webofthings.com/energievisible/.

[229] EPCglobal architecture framework standards. http://www.gs1.org/gsmp/kc/epcglobal/architecture.

[230] EPCglobal discovery services standard (in development). http://www.gs1.org/gsmp/kc/epcglobal/discovery.

[231] European environment agency: Maps and graphs. http://www.eea.europa.eu/data-and-maps/figures#c15=all&c5=&c9=&c0=15&b_start=0.

[232] Extended environments markup language: EEML. http://www.eeml.org/.

[233] Feel, act, make sense - sen.se. http://open.sen.se/.

[234] Fosstrak - open source software for track and trace. http://www.fosstrak.org/.

[235] Fosstrak EPCIS - EPCIS webadapter home page. http://www.fosstrak.org/epcis/docs/webadapter-guide.html.

[236] GlassFish: open source application server. http://glassfish.java.net/.

[237] Google maps. http://maps.google.com.

[238] Google visualization API reference - google chart tools - google code. http://code.google.com/apis/chart/interactive/docs/reference.html.

[239] Google web toolkit. http://code.google.com/intl/en/webtoolkit/.

[240] Grizzly NIO web server. http://grizzly.java.net/.

[241] HousingMaps. http://www.housingmaps.com.

[242] IETF working group for constrained RESTful environments. http://tools.ietf.org/wg/core/.

[243] Internet of things - ThingSpeak. https://www.thingspeak.com/.

[244] IPSO alliance: Enabling the internet of things. http://ipso-alliance.org/.

[245] iui - web UI framework for mobile devices - iOS, android, palm, and others. http://code.google.com/p/iui/.

[246] The java community Process(SM) program - JSRs: java specification requests - JSR# 244 (jdo). http://www.jcp.org/en/jsr/detail?id=243.

[247] Java ME (mobile and embedded edition). http://www.oracle.com/technetwork/java/javame/index.html.

[248] JAX-RS standard for RESTful web services (JSR 311). http://jsr311.java.net/.

[249] Jersey JAX-RS reference implementation for building RESTful web service. http://jersey.java.net/.

[250] jQuery: the write less, do more, JavaScript library. http://jquery.com/.

[251] JSR-000220 enterprise JavaBeans 3.0 - final release. http://jcp.org/aboutJava/communityprocess/final/jsr220/index.html.

[252] kSOAP 2 WS-* for resource constrained devices. http://ksoap2.sourceforge.net/.

[253] Metering portal: Information for metering professionals. http://metering.com.

[254] Microformats. http://microformats.org/.

[255] Mobile JavaScript framework for developing HTML5 web applications. http://www.sencha.com/products/touch/.

[256] Mongoose - easy to use embedded web server. http://code.google.com/p/mongoose/.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0356

[257] NorhTec - products. http://www.norhtec.com/products/mcsr/index.html.

[258] OpenPICUS is an open source embedded platform for smart sensors and the internet of things. http://www.openpicus.com/cms/.

[259] OpenWrt: linux distribution for embedded devices. https://openwrt.org/.

[260] OSGi alliance. http://www.osgi.org/Main/HomePage.

[261] Phidgets prototyping RFID reader. http://www.phidgets.com/products.php?product_id=1023.

[262] Plogg: wireless energy sensor networks. http://www.plogginternational.com/.

[263] ProgrammableWeb - mashups, APIs, and the web as platform. http://www.programmableweb.com.

[264] pubsubhubbub - a simple, open, web-hook-based pubsub protocol & open source reference implementation. http://code.google.com/p/pubsubhubbub/.

[265] RDFa in XHTML: syntax and processing. http://www.w3.org/TR/rdfa-syntax/.

[266] Real-Time open data web service for the internet of things - pachube. http://pachube.com/.

[267] REST binding project at AUTO-ID labs at MIT. http://autoidlabs.mit.edu/CS/content/OpenSource.aspx.

[268] Restlet - RESTful web framework for java. http://www.restlet.org/.

[269] Roving networks wifi modules. http://www.rovingnetworks.com/.

[270] ruote - open source ruby workflow engine. http://ruote.rubyforge.org/.

[271] Server-Sent events. http://dev.w3.org/html5/eventsource/.

[272] Size of wikipedia. http://en.wikipedia.org/wiki/Wikipedia:Size_of_Wikipedia.

[273] SunSPOT - embedded development platform. http://www.sunspotworld.com/.

[274] ThingWorx the 1st application platform for the connected world. http://www.thingworx.com/.

[275] Travel - guides - the new york times, supporting microformats. http://travel.nytimes.com/.

[276] Ubuntu cloud. http://www.ubuntu.com/business/cloud/overview.

[277] Unbeatable JavaScript tools - the dojo toolkit. http://dojotoolkit.org/.

[278] URIs, URLs, and URNs: clarifications and recommendations 1.0. http://www.w3.org/TR/uri-clarification/#contemporary.

[279] VirtualBox. http://www.virtualbox.org/.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0357

[280] VMware virtualization software for desktops, servers & virtual machines for public and private cloud solutions. http://www.vmware.com/.

[281] Webnergy - a project to web-enable smart meters. http://code.google.com/p/webnergy/.

[282] The WebSocket API. http://dev.w3.org/html5/websockets/.

[283] Weebiz: Business community. https://weebiz.com/.

[284] Wikipedia – comet programming. http://en.wikipedia.org/wiki/Comet_(programming).

[285] Wikipedia – reflection in computer programming. http://en.wikipedia.org/wiki/Reflection_(computer_programming).

[286] WS4D web services for devices. http://www.ws4d.org/.

[287] Yahoo pipes mashup editor: Rewire the web. http://pipes.yahoo.com/pipes/.

[288] Yaler - access small devices from the web. http://yaler.org/.

[289] Yahoo search blog: We now support microformats. http://www.ysearchblog.com/2006/06/21/we-now-support-microformats/, June 2006.

[290] Marking up products for rich snippets - webmaster tools help. http://www.google.com/support/webmasters/bin/answer.py?answer=146750, March 2011.

[291] Tim Berners-Lee. Keynote at WWW 2009: Twenty years: Looking forward, looking back. http://www2009.eprints.org/212/1/index.html, April 2009.

[292] Tantek Celik. Geo microformats. http://microformats.org/wiki/geo, November 2009.

[293] Tantek Celik, Ali Diab, Ian McAllister, John Panzer, Adam Rifkin, and Michael Sippey. hReview 0.3 microformats. http://microformats.org/wiki/hReview, February 2006.

[294] Tantek Celik and Brian Suda. hCard 1.0 microformats wiki. http://microformats.org/wiki/hCard, May 2011.

[295] Rory Cellan-Jones. BBC - dot.life: Things that tweet. http://www.bbc.co.uk/blogs/technology/2009/06/things_that_tweet.html, June 2009.

[296] F. Dawson and T. Howes. vCard MIME directory profile, IETF memo. http://www.ietf.org/rfc/rfc2426.txt, September 1998.

[297] Paul Lee and Jay Myers. hProduct microformat draft specification. http://microformats.org/wiki/hproduct, June 2010.

[298] Lukas Naef. ClickScript: easy to use visual programming language. http://clickscript.ch/site/home.php.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0358

[299] OpenSocial and Gadgets Specification Group. OpenSocial RESTful protocol specification v0.9. http://www.opensocial.org/Technical-Resources/opensocial-spec-v09/REST-API.html, April 2009.

[300] G. J. Rothfuss. Official google maps API blog: Microformats in google maps. http://googlemapsapi.blogspot.com/2007/06/microformats-in-google-maps.html, July 2007.

[301] Alex Williams. SOAP is not dead - it's undead, a zombie in the enterprise. http://www.readwriteweb.com/enterprise/2011/05/soap-is-not-dead---its-undead.php.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0359

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0360

Curriculum Vitae: Dominique Guinard

Personal Data

Date of Birth	February 27, 1981 in Fribourg
Citizenship	Swiss

Education

2007–2011	*ETH Zurich*, Ph.D. Student in Pervasive Computing (Department of Computer Science), Zurich Switzerland
2010–2011	*MIT*, Visiting Researcher at the MIT Auto-ID Labs, Cambridge, USA
2006–2007	*University of Lancaster*, Visiting Graduate Student in Ubiquitous Computing (Master thesis), Lancaster, UK
2005–2007	*Universities of Fribourg and Bern*, M.Sc. in Computer Science, Fribourg & Bern, Switzerland
2002–2005	*University of Fribourg*, B.Sc. in Computer Science and Business Management, Fribourg, Switzerland
1998–2001	*Collège de Gambach*, Baccalauréat in Business Administration (High school), Fribourg, Switzerland

Employment

2007 – 2011	Research Associate, *SAP Research*, Zurich, Switzerland
2008 – 2011	Research Assistant, *Department of Computer Science, ETH Zurich*, Zurich, Switzerland
2007 – 2008	Research Associate, *ETH Auto-ID Labs*, Zurich, Switzerland
2005 – 2007	External Scientific Collaborator, *Software Engineering Group, University of Fribourg*, Fribourg, Switzerland
2005 – 2006	Teaching Assistant, *Pervasive and Artificial Intelligence Group, University of Fribourg*, Fribourg, Switzerland
2002 – 2006	Professional Trainer in Web Development, *Ecole-Club Migros*, Fribourg, Switzerland
1999 – 2006	Co-founder, Project Manager, *Spoker and GMIPSoft*, Fribourg, Switzerland
2005 – 2005	Intern at the Software Practice, *Sun Microsystems*, Gland, Switzerland
1998 – 1999	Web Developer, *Dartfish*, Fribourg, Switzerland

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0361

Appendix Hall 05

Scheduled Maintenance: On Tuesday, 1 October, IEEE *Xplore* will undergo scheduled maintenance from 1:00-5:00 PM ET (1700-2100 UTC). During this time, there may be intermittent impact on performance. We apologize for any inconvenience.

IEEE.org | IEEE *Xplore* | IEEE SA | IEEE Spectrum | More Sites Subscribe Donate | Cart | Create Account | Personal Sign In

IEEE *Xplore*® Browse My Settings Help Institutional Sign In IEEE

All

ADVANCED SEARCH

Conferences > 2012 First International Work...

Establishing a smart grid node architecture and demonstrator in an office environment using the SOA approach

Publisher: IEEE Cite This PDF

Dagmar Koß ; Denis Bytschkow ; Pragya Kirti Gupta ; Bernhard Schätz ; Florian Sellmayr ; Stef... All Authors

19 Cites in Papers

359 Full Text Views

Abstract

Document Sections

I. Introduction

II. Living Lab

III. Architecture

IV. Implementation

V. Experiences

Show Full Outline

Authors

Figures

References

Citations

Keywords

Metrics

Abstract:
The introduction of low-cost renewable energy production, e.g., by photovoltaic, has turned classical grid nodes like homes and offices in prosumers, taking an active role in smart energy systems by merging home-automation and energy production functionality. However, to become a self-balancing element of a stable smart grid, supporting the energy-aware cooperative production, storage, and consumption, a scalable software is needed, tailored for smart micro grids and their integration in large-scale systems. In the following, the implementation of a layered SOA-based distributed architecture is presented, that provides open interfaces simplifying the plug-and-play of hardware and software components.

Published in: 2012 First International Workshop on Software Engineering Challenges for the Smart Grid (SE-SmartGrids)

Date of Conference: 03-03 June 2012

DOI: 10.1109/SE4SG.2012.6225710

Date Added to IEEE *Xplore*: 28 June 2012

Publisher: IEEE

Conference Location: Zurich, Switzerland

ISBN Information:

I. Introduction
The increasing integration of renewable energy from private households and companies, which are connected to the medium and low voltage level starts to change our existing energy system drastically. New challenges in the energy system do not only include the balancing of volume energy production and consumption all control infrastructure

Sign in to Continue Reading

Authors

Figures

More Like This

Methods and Algorithms for the Interaction of Residential Smart Buildings with Smart Grids
2015 IEEE 13th International Conference on Embedded and Ubiquitous Computing
Published: 2015

Distributed Real-Time HVAC Control for Cost-Efficient Commercial Buildings Under Smart Grid Environment
IEEE Internet of Things Journal
Published: 2018

Show More

References

Citations

Keywords

Metrics

IEEE Personal Account

CHANGE USERNAME/PASSWORD

Purchase Details

PAYMENT OPTIONS

VIEW PURCHASED DOCUMENTS

Profile Information

COMMUNICATIONS PREFERENCES

PROFESSION AND EDUCATION

TECHNICAL INTERESTS

Need Help?

US & CANADA: +1 800 678 4333

WORLDWIDE: +1 732 981 0060

CONTACT & SUPPORT

Follow

About IEEE *Xplore* | Contact Us | Help | Accessibility | Terms of Use | Nondiscrimination Policy | IEEE Ethics Reporting | Sitemap | IEEE Privacy Policy

A not-for-profit organization, IEEE is the world's largest technical professional organization dedicated to advancing technology for the benefit of humanity.

© Copyright 2024 IEEE - All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies.

Establishing a Smart Grid Node Architecture and Demonstrator in an Office Environment using the SOA Approach

Dagmar Koß, Florian Sellmayr, Steffen Bauereiß, Denis Bytschkow, Pragya Kirti Gupta, Bernhard Schätz
fortiss GmbH, Technical University of Munich
Munich, Germany
{*koss,bytschkow,gupta,schaetz*}*@fortiss.org,* {*sellmayr,bauerest*}*@in.tum.de*

***Abstract*—The introduction of low-cost renewable energy production, e.g., by photovoltaic, has turned classical grid nodes like homes and offices in prosumers, taking an active role in smart energy systems by merging home-automation and energy production functionality. However, to become a self-balancing element of a stable smart grid, supporting the energy-aware cooperative production, storage, and consumption, a scalable software is needed tailored for smart micro grids and their integration in large-scale systems. In the following, the implementation of a layered SOA-based distributed architecture is presented, that provides open interfaces simplifying the plug-and-play of hardware and software components, allows the runtime adaption of its reactive behavior to incrementally enhance its intelligence, and uses an interaction paradigm supporting an open and distributed form of cooperation.**

***Keywords*-Smart Grid, Demonstrator, Smart Energy, Layered Architecture, Office Automation, System Architecture, SOA, OSGi, ActiveMQ**

I. Introduction

The increasing integration of renewable energy from private households and companies, which are connected to the medium and low voltage level, starts to change our energy system drastically. New challenges in the energy system include not only the balancing of volatile, largely distributed, small-volume energy production and consumption, but also a shift of centralized overall control infrastructure to more localized and decentralized approaches. This requires new energy systems, which will enable decisions and the management of available energy production and distribution on the local level.

This paper focuses on establishing new system architectures and their implementation by providing a realization in form of a demonstrator for an element in distributed energy systems, forming an intelligent node in a smart grid infrastructure environment. Smart grid nodes include energy consumption, production, buffer capacity, sensors and ICT components. In order to achieve smart grid requirements like integration and independence of electrical devices including plug and play capabilities, scalability, distributable, robustness and self-recovery, the developed smart node architecture is designed maximal flexibility. Besides the control of heterogeneous devices, the architecture also covers the integration of sensors, in order to enable control mechanisms for energy efficiency and the possibility to run a wide variety of different application scenarios, providing a *Living Lab* to evaluate the applicability of the architecture.

Our smart grid node demonstrator provides the needed infrastructure to integrate several heterogeneous devices in an office space environment (such as smart meters, home automation units, sensors, solar panels, batteries, air-condition etc.). The smart node enables metering/monitoring, storing, processing, and analyzing the status and data of individual devices, aggregating it into the status of the overall smart grid node. The demonstrator also includes methods to decide on and perform actions to achieve a better energy efficiency (e.g., turning off unused lights). Besides basic energy saving the *Living Lab* can be used to demonstrate methods, which are currently investigated in the smart home research community. This includes optimization and control of appliances (like dish washer, washing machine, refrigerator, smart home devices etc.), which can use energy at times when the energy prices are lower or can be rescheduled to another time to smooth unexpected energy peaks, as it was shown in [1].

One of the main challenges to establish a demonstrator that offers lots of interoperability was the integration of different devices. Most of the devices in the *Living Lab* provide different interfaces and data structures. For example: The home automation devices are communicating via EnOcean [2], messages while the smart meters have either a built-in web front-end (IPswitch [3]) or communicate via modbus (PacSentron[4]). Right now, there is not yet a common interface to access the devices data that can be used to establish a smart building, without the need for components that translate between different protocols.

The developed *Living Lab* enables the collection of real life data, which is important to find out what kind of data is needed, how much data is produced by the different devices and what monitoring frequency is required for the system to control the different devices. The collected data will help to get a deeper understanding about the data processes in smart nodes, which will be needed in the future. Our smart grid node *Living Lab* will also provide the possibility to connect to other smart grid demonstrators.

This paper is structured as follows: the next section will provide a short overview of the *Living Lab* demonstrator

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0365

environment. Afterwards, the developed smart node architecture is described in section III, while section IV demonstrates its implementation. The paper continues with experiences in section V and finishes with a conclusion and an outlook in Section VI.

II. Living Lab

A. Demonstrator Hardware Plattform and Environment

The goal of *Living Lab* smart grid node is the definition of a flexible prosumer node architecture, merging energy management and home automation aspects and facilitating the integration of different device types, providing an extensive set of data. Furthermore, providing a layered, distributed architecture, its control functionality can be easily extended. As a *Living Lab*, the demonstrator is implemented as part of the regular office space of fortiss. The general installation and the existing energy and data flow is shown in figure 1.

As the core elements of the technical platform, four major categories of devices are installed. The first group of devices is related to the energy production and storage. A 5 kWh photovoltaic system is installed on the roof of the building. In addition to the solar panels, a battery system including an intelligent charge controller provides sufficient capacity to supply a set of offices with energy running in island mode (i.e., decoupled from the metropolitan grid). The data from the photovoltaic system is gathered by a data logging device and provided via web interface.

To support usage scenario of a work-place setting, an office room and a meeting room are part of the *Living Lap*. Both rooms are equipped with devices of the second category: actors and sensors of the EnOcean [2] familiy, which represents the home automation part of our *Living Lab*. EnOcean devices are wireless and powerless home automation components. The sensors, which are installed within our office space, are: Temperature sensors (indoor and outdoor), occupancy sensors, window sensors and lightness sensors (indoor and outdoor), switches for the blinds and lights. To control devices like blinds or lights, actors were installed, which receive the signals either from the switches itself, or from an EnOcean gateway via Ethernet.

The third category of installed devices are smart meters. One Siemens PacSentron 4200 [4] smart meter is installed in the office space and measures both demonstrator rooms. Another PacSentron is installed in the server room and measures the consumption of two air conditioner units. For a fine-grained monitoring of consumption data, three further PacSentrons are installed to measure all other office rooms, which are currently not integrated into the demonstrator. All PacSentrons are connected via Ethernet (ModBus via Ethernet) to the office LAN and answer to a gateway, translating the ModBus data from the meters into IEC 61850 [5] conform data.

The category of smart meters also includes an IPswitch meter [3]. This IPswitch is comprises three electric meters which measure the energy consumption of high-end computing servers (including a switch and a big data-storage device) and the interruptible power supply (UPS) backup system. A three phased meter is connected to one UPS, two other two phased meters are connected to the servers. The IPswitch provides an Ethernet interface to the local network and has a built-in web server, which presents the current energy consumption on a web page. Additionally, there is one electric meter from EnOcean. It measures the energy consumption of the lights in the meeting room.

The fourth category of devices is an air conditioner control management system for the two air conditioner units in the server room. The data (like temperature, cooling hours, status, etc.) of the air conditioner units is distributed over Ethernet via a web-interface, which allows their monitoring and control.

All devices of the *Living Lab* are connected via Ethernet. This Ethernet serves as communication backbone for the developed architecture, described in Section III. The developed software, which collects and monitors the data and controls the devices is discussed in Section IV.

B. Related Work

In the field of smart homes, substantial research including the implementation of demonstrators has been performed [1] [6]. It includes integration of intelligent household appliances, solar panels, a combined heat and power plant (CHP) and an e-car battery into a smart home. The demonstrators investigates the management of the energy demand in future smart grid, implementing an observer/controller architecture to achieve the goal of adaptation and automatic control. The architecture was chosen from the context of organic computing [7][8], whereas our focus is more based on service orientation to build up a flexible, distributed platform of COTS technology for further research activities, facilitating device integration and system interoperability.

In the home automation domain, a wide variety of smart devices and corresponding approaches for load-balance & price-efficient techniques inside a residential building have been investigated, e.g., [9] [10] [11]. There has been research going on for automation of smart office buildings [12] [13]. [14] points to the effort in building energy efficient office infrastructure based on dynamic pricing and operation scheduling using predefined policies, implemented by a set of rules that must hold for the operation of each associated device. In this paper, we propose policies for the operation of devices, but do not coupled those policies with the devices. The rule system component – described in detail in SectionIV – supports user-defined rules, which can be changed at run-time.

On the architectural level, there have been attempts to use SOA for smart homes based on OSGi capability. [15] proposes a SOA architecture for smart homes based on OSGi with Mobile-Agent technology. Mobile Agents (MAs)

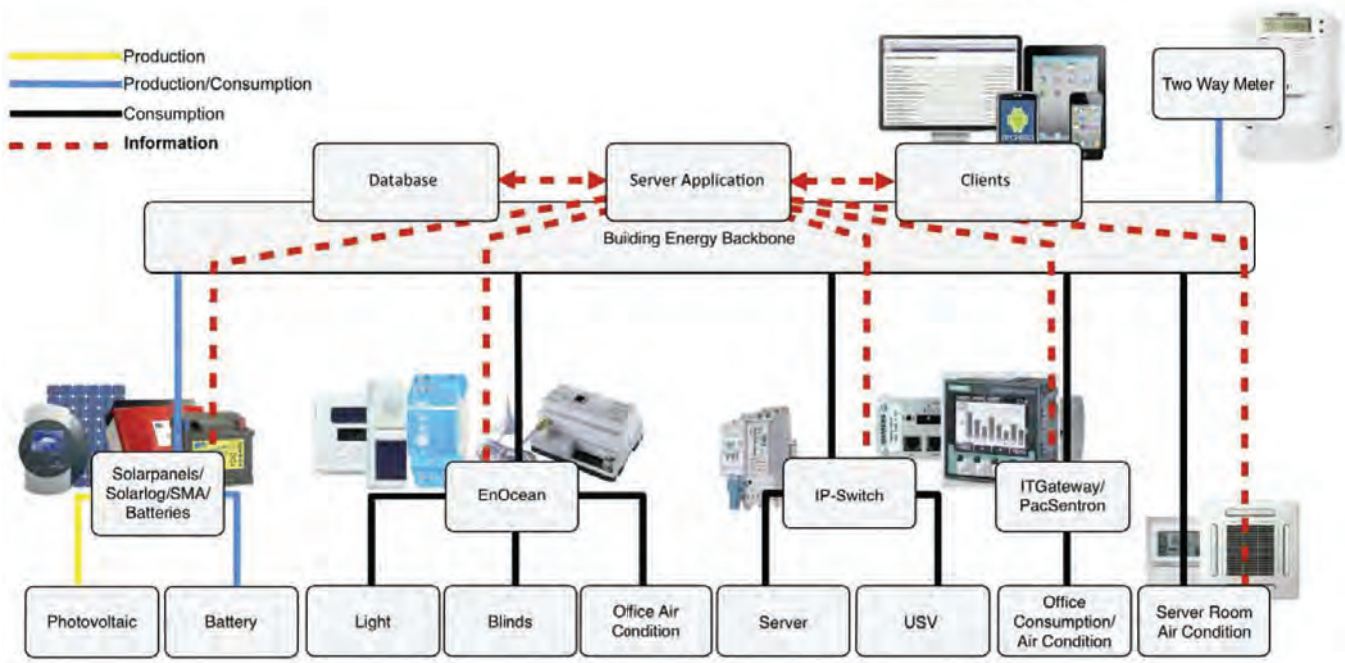


Figure 1. The Living Lab Demonstrator including information and energy flow

are the software modules, which support the automatic interaction between various components of the system. As pointed out in there, the inability of the MAs to handle unexpected exceptions makes them less applicable. [16] propose a similar approach as presented here. That approach emphasized on using UDDI and SOAP via XML, with the disadvantage that UDDI has inefficient mechanism to handle service publishing and discovery. In our proposed architecture, we use OSGi as principle runtime environment and Apache ActiveMQ for the interaction between the services.

III. Architecture

To build up the demonstrator, a layered architecture was developed, specifically providing functionality for smart grid nodes. The layered architecture reduces the complexity of the system, while supporting a large set of heterogenous devices (with different interfaces installed in different locations). It enables the user to easily integrate new devices into the system, support the communication, and process large amounts of data. For the support of decision making and control of the system, a rule system was integrated.

A. Layered Architecture

A smart grid node architecture needs to combine two goals. It has to work in a heterogeneous hardware landscape while providing a generalized interface for usage in higher level domains such as data-analysis, event-processing and control applications. To combine these two goals the architecture can be divided into layers. We use three layers: A device layer, a communication layer, and a processing/control layer, which is shown in figure 2.

The device layer deals translates the device specific protocols into a device independent internal protocol, using an event-based communication paradigm. This encapsulation facilitates the implementation of higher-level functionalityby abstracting all hardware-dependent details (e.g. vendor-specific protocols). Moreover, extending the system to a new set of devices is supported by developing a wrapper that implements an interface (see IV-B). By providing such a wrapper, adding new hardware to the node can be performed in a plug and play-fashion.

To integrate the heterogeneous components, a service bus provides the necessary communication layer. It enables components to interact either in a one-to-one or publish/subscribe manner depending on the use-case: One-to-one communication enables the interaction with a specific endpoint (e.g. controlling a specific device), while a publish/subscribe mechanism enables to broadcast events, which are useful for a set of components (subscribers) within the system (e.g. sensor-data to be sent to a decision-making as well as a logging component). This layers also provides the mechanims to connect the node to other systems including a billing system or other smart grid elements.

On top of the communication layer, high-level services provide more complex functionality. It includes a service for persisting all events in a database for analysis. The rule system as a more advanced service provides a decision system for the control of the node using a set of predefined

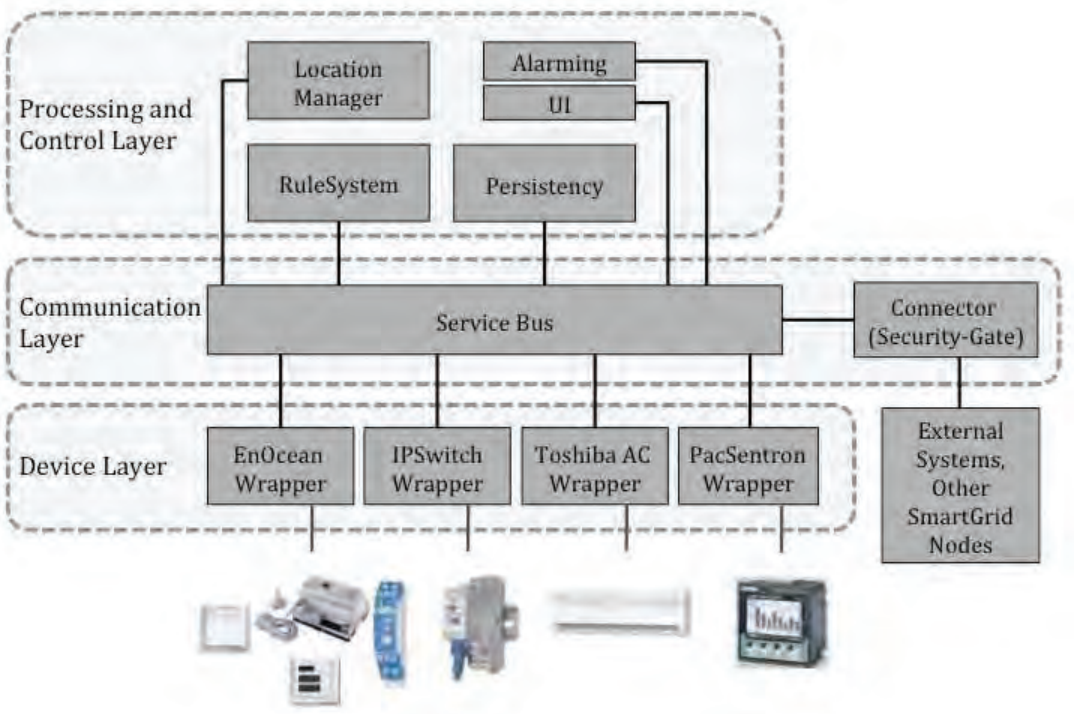


Figure 2. Architecture of the fortiss *Living Lab* demonstrator

strategies (e.g. to turn off certain devices when a smart meter reports high energy costs).

B. Design Principles

1) Event-driven communication: Sensor hardware provides its information either pull based (e.g. displayed on an HTML page like IPswitch) or pushed based (e.g. EnOcean-Gateways which stream EnOcean messages into the system as they arrive). To provide a consistent and simple interface for all devices, one of those approaches had to be chosen for the system.

In a push-based architecture, poll-based devices would be polled by the wrappers in the device layer and an event would be generated after each poll or under certain conditions (i.e. the difference between the current sensor value and the last was above a certain threshold). In a pull-based architecture the device layer components for push-based devices would need to cache the most recent events from all their devices and provide them until being polled.

The push-based (or event-driven) approach has several advantages for our purpose:

- New sensor-data is propagated almost instantaneous and without the overhead produced by polling
- No effort has to be made to cache push-based devices information for polling
- No additional polling logic is required for implementation of higher-level components

One potential drawbacks of such a design is the fact that the system does not provide an inherent sense of its current state, requiring higher level components to track such information. However, the persistency service of the proposed architecture responsible for storing events proved to be sufficient for the implemented functionality, including all kind of user interfaces and other higher-level services as the rule system described in see IV-D3, which requires a higher performance that could not be achieved by polling either.

2) Service Oriented Architecture: The most important aspect of SOA is the service: A service is typically a component within the system that provides a clearly defined, single functionality (such as "long term persistence of sensor data") and encapsulates all logic and data necessary for this task [17].

Such a component must not to be confused with a technical service (e.g. access to a database): A SOA should be technology-agnostic, leaving technical details to the service implementation.

To access data or logic represented by a service, the service provides a high-level service contract and an interface that provides all information, which is necessary to work with it. This service interface reflects application domain principles, not technical ones (e.g., *"give me the most recent sensor value for sensor42"*, not *"execute SELECT ..."*). This facilitates to interact with services without knowledge about their internal structure and to replace a service imple-

mentation without interfering with other components [18] (4.3.2 and 5.1).

Services can provide basic functionalities (such as exposing a certain piece of hardware via a clearly defined interface, providing historic sensor data or information about relationships between devices) or use other services to provide more high level functionalities such as controlling the node by consuming the services of wrappers and components providing historic values or information on the relationships between devices.

Note that the terms of "basic" or "higher level" services do not necessarily match the architecture layers: For example, a persistence service is a "basic" service since it does not consume any other service but is nevertheless situated in the highest layer since it provides information for processing functionality.

IV. Implementation

The following section provides an overview of how the design mentioned before was implemented in the *Living Lab* demonstrator. We describe the most important technical infrastructure before providing more detail about the interfaces that describe sensor data and actor control information. The last subsection outlines in more detail the components that provide the abstract services described above.

A. Technical Infrastructure

1) Service-Bus: The current implementation of the communication layer uses Apache ActiveMQ [19] as a service-bus implementing the Java Message Service standard [20] and utilizing SOAP for message encoding. This makes all components accessible in a web-service-like manner and provides one-to-one as well as the publish-subscribe communication mechanisms as mentioned above.

By using Enterprise Integration frameworks such as Apache Camel [21] it is also possible to integrate external components that were not built for a service-bus architecture (as a proof-of-concept, an off-the-shelf alarming system developed by an external vendor was integrated using this technique).

2) Component Platform: While the system is prepared to run in a highly distributed environment, hosting it on a single platform makes it much easier to maintain. It currently runs in an OSGi [22] environment as a component platform, which enables us to activate, deactivate or deploy system modules easily, providing a platform on which new components can be integrated in a plug and play fashion. While [15] uses an OSGi-based agent-oriented approach for adding new devices, the *Living Lab* does not rely on specific extension of the OSGi platform to keep the design as flexible and technology independent as possible.

While implementation as a distributed system are not fully exploited at this point with all hardware and software components locally integrated, in a more advanced scenario with a distributed collection of devices or a microgrid of smart nodes can run their own rule systems.

B. Interfaces

The core services of the *Living Lab* demonstrator are the services for processing sensor data and controlling the actors.

The first service is provided by the communication layer, which allows the wrappers in the device layer to trigger events, i.e., messages indicating updated sensor values. The wrappers in the device layer provide a service for processing commands, i.e. processing messages to control a certain device. Interfaces to both services are designed symmetrically.

At the moment, established interface standards exist only for a subset of the functionality, which is required for a smart grid node. E.g., the IEC 61850 [5] based interface describes aspects of power consumption very well, but is much less suitable for other sensor families, such as temperature sensors, and does not address controlling devices at all. Thus, an interface specification had to be defined to suit all these requirements. In principle, the definition of such an interface can follow two design paradigms: Generalized, with a very basic and simple interface, or specialized, with a lot of domain-knowledge modeled into the interface.

A specialized approach has the advantage of a more expressive semantics, that can be used to clearly classify an event (e.g. *"light-switch 3 changed to state down"*). However, such an interface definition requires a mature domain model. Since domain engineering was not the primary objective of this research, a generic approach was chosen.

Sensor data is described as a simple event, which is based on primitive data types:

- A *double*-event for measurements including a unit and a value for precision of information such as temperature, position of blinds, energy consumption, lightness…
- A *boolean*-event for on/off type events such as state of a button, or true/false events such as window closed/opened, room occupancy true/false…
- A *toggle*-event for events without any semantics, e.g. pushing a stateless button

Such an interface definition covers all essential event classes needed in a smart grid node setting, and can be easily extended by adding a more domain-specific model along the lines of IEC 61850 on top of it.

Events corresponding to commands to devices are defined in a similar way. They also include additional attributes to delay or group events. E.g., a *boolean* command to a light would switch a light on or off, while a *double* command to the blinds could trigger the actor to move the blinds to 25%.

Finally, state change events are provided to describe changes in the system. They can, e.g., used to describe 'virtual commands' to keep track of actors that are triggered outside of system control. For example, in case a light switch is directly coupled to light-actor, the actor is triggered

directly by the switch without rule system involvement. Thus, the system can record the switch-event as well as the triggered action with its *boolean*-event, bringing the system state back in sync with the real world. This mechanism with controlled and observed actions and state changes can be used to make sure basic functions can still be controlled manually in case of a system malfunction or maintenance.

To announce the existence of new devices, the interface also defines a *newDevice* event. Thus, new wrappers only need to be configured to know the systems service-bus and send such an event to be incorporated into the system. Adding a discovery mechanism (which is currently not implemented but possible with the underlying service bus implementation), prepares the system for complete plug and play (always assuming the device does speak the described protocols or a wrapper was created for it).

C. Device Addressing

In a pure service-oriented approach, every single device (every light, button, air conditioner,...) would expose and consume services and thus be addressed individually on the service-bus. This approach was found to be highly inefficient in early prototype benchmarks since current frameworks were not designed for such a large number of endpoints.

Thus, devices in the system are addressed with a combination of their wrappers endpoint-address (e.g. *EnOcean1*) and a device-ID that uniquely identifies the device within the wrapper. Such a device-ID could either be the devices real ID as defined by the vendor (e.g. *010088C8*) or the ID is defined by the user when configuring the device (e.g. *officelight152_1030*). Note that IDs are primarily of technical nature. More human readable names can be defined in a meta-data storing component (see section IV-D).

D. Components

1) Wrappers: In the lowest layer, the wrappers act as a source of information for the system. The amount of information (data points) pushed to the system as events depends on the device capabilities and the implementation of the wrapper. Some devices periodically push their data to connected listeners, whereas others send data upon changes only. The wrapper can listen for any data and forward this information as events. If necessary periodically pushed information can be held back by the wrapper and forwarded upon changes only. Another device type has to be polled by the wrapper. When designing the wrapper a polling-rate suitable for the type of the device should be chosen. In the *Living Lab* wrappers for EnOcean, IPswitch, air conditioner control and PacSentron are collecting data from the connected devices. EnOcean follows the push approach, while the other three wrappers are polling their devices.

2) LocationManager: To be able to control the smart grid node effectively, additional metadata about all devices, their relationships and their environment is necessary for easier configuration and maintenance of the system (also in regard of the rulesystem see section IV-D3). Such metadata includes a unique device name, device type and physical location. Relationships between devices are associations influencing for the system behavior (e.g. *"EnOcean1?010088C8 must switch toshibaAC?srv1"* or *"toshibaAC?srv1 is in datacenter"*) or model connections in the physical world the system must be aware of (e.g. *"EnOcean1?01003F84 directly wired to EnOcean?officelight225_1030"*). Basically, part of the information could be hard-wired via the individual wrappers on the device layer. However, the set of metadata required also contains information, for which the wrapper is not responsible for, such as human-readable names and relationships between devices of different classes and vendors (e.g. *EnOcean sensor triggers air conditioner*).

To avoid dispersion of information over several components and to support a more structured represenation of information (e.g., grouping of devices to rooms, floors, etc.), the responsibility for such metadata was delegated to a separate component, the *LocationManager*. It uses the graph database neo4j [23] to model all associations between devices, rooms and possibly different buildings in a graph. Additionally neo4j offers a key-value store for every node, used for storing the device-specific information. Figure 3 shows how the devices described above can be modeled in the LocationManager.

3) Rulesystem: To control the smart grid node, in the approach presented here, a set of rules (i.e., a set of actions executed upon the satisfaction of their triggering conditions). In the current implementation, the *Rulesystem* component registers with the service-bus to receive all events issued in the system. Upon receiving one of the events the component uses the JBoss Drools rule engine [24] to process the event and act accordingly. A simple rule for turning on the air conditioner in case the temperature rises above 25 degrees Celsius is given in Figure 4. This rule controls every air condition system that has been connected to a certain temperature sensor, ensuring that air condition is turned on as soon as a sensor sends a temperature event with a value above the defined threshold. It relies on the *LocationManager* component to identify which air condition unit will be activated, using the information about the devices and their relationships. This enables the user to write rules in a generalized fashion, avoiding to duplicate similar device-specific rules for each applicable combination of sensors and actors. This reduces the complexity of the rule system, which would result in an bloated rule base hard to maintain, slowing down the system, and increasing the probability of contradictory rules.

4) Persistence: The *Persistency* component registers with the service-bus as an event handler. All information from received events is written to a relational database and can be queried using the services offered by the persistency component. The component represents the central data collection

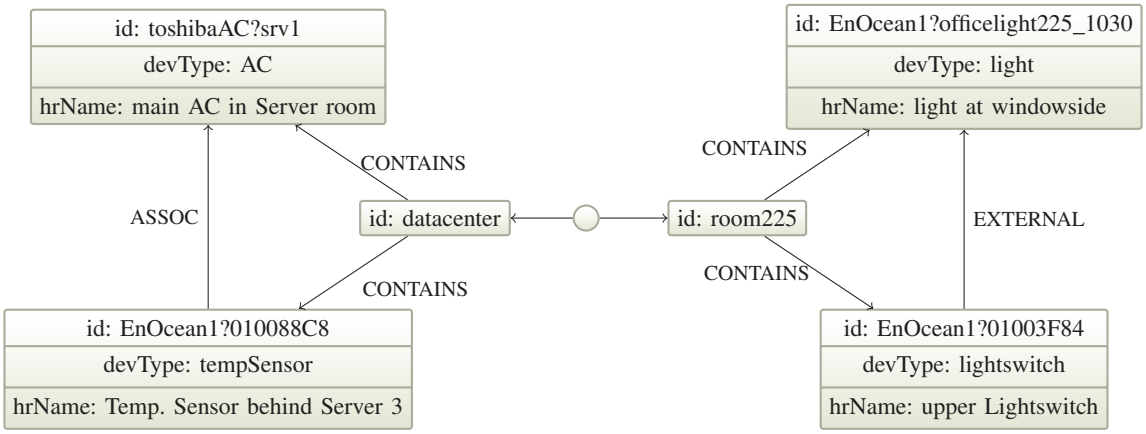


Figure 3. Schematic example for a graph stored in the LocationManager

```
rule "Simple AC/Temp Example"
when
//Check for events
   DoubleEvent(
      $val: value,
      "Celsius",
       $org : origin
      ),
   $val: 25.0;
then
//issue command for every device associated
//with the events origin
for (Device d :
   locationManager.getAssociatedDevices($org)){
   if (d.devType == "AC"){
   //Turn ON command issued here
   commandUtil.setBoolean(true, d);
   }
}
end
```

Figure 4. Example of a rule, turning on the air-conditioning when the temperature raises above a certain level.

and accumulation. The collected data features an average of 14.000 data points collected during a single day in our *Living Lab*. This large amount of data is due to a detailed generation of events in the current setup. All events from push-based devices are passed on as events, while wrappers that require polling are typically generating a full set of events when being polled (e.g. every 10 seconds). The amount of data can be reduced significantly by filtering duplicate events within a certain time frame or reducing the overall polling frequency.

V. Experiences

A. Hardware Pitfalls

When implementing the wrappers for the device layer we repeatedly experienced that some hardware is not yet prepared to be integrated into a heterogeneous system or even integrated with other software at all: Some hardware vendors do not provide any defined API, making it necessary to implement fragile web-scraping solutions to get the necessary data. Other vendors appear to provide such interfaces but do not test or define them very well: We had to deal with firmware that was unsuitable for some common use-cases, software that became unstable when running over longer periods of time and gaps, errors or inconsistencies in protocol or interface-specifications.

Clearly defined standards would certainly help to make building automation devices more interoperable.

B. Amount of Gathered Data

As mentioned in section IV-D4, the system gathers several millions of data points in a few weeks. While this is not an issue in terms of storage, impacts on query-performance can already be experienced on the demonstrator. Here, more efficient ways to aggregate and persist the data need to be added. To that end the necessary degree of detail of the sampled data has to be investigated, needed for the application scenarios. Here, strategies with different time windows with different levels of details for different sets of data including corresponding aggregation techniques must be provided.

C. Efficiency of Communication Protocols

We chose SOAP as a communication protocol primarily for its flexibility and ease of use when developing additions to the system independent of the technical details. However, SOAP is by far inferior to other message encoding methods in terms of efficiency due to the costs of XML marshaling and remarshaling [25].

At this point, such costs are not yet a problem for the basic system-functions, such as gathering sensor data or control, but they are already visible to the user in the form of a sluggish response, e.g. when metadata for a number of devices has to be pulled from the LocationManager via the Service Bus.

Thus, it should be investigated whether the high flexibility, which SOAP provides, is really necessary and can be justified for all requirements.

D. Basic Data-Type Interface

The lack of a domain-specific data model providing a more specific semantic interpretation of events (see section IV-B) did not pose a problem concerning the incremental extension of the node architecture. Different extensions were implemented on top of the existing architecture by different user groups with only minimal training efforts.

Our experience shows that within the context of a prosumer node the need for additional semantic information about data (e.g. *"the event came from a light switch"*) typically correlates with the need for additional metadata about the device itself (e.g. a location or a connection to other devices, *"which devices are connected to the light switch"*), which in turn is provided by the Location Manager component described in Section IV-D. This component proved sufficient in our setting to provide the necessary semantic meaning for basic data-type events (such as the type of device an event originated from). However, for the overall grid architecture, a more elaborate data model is likely to be necessary.

VI. Conclusions / Outlook

This paper introduces an architecture for smart grid nodes using a layered, service-oriented approach. The implementation in form of a *Living Lab* demonstrator has sown the suitable of three distinct layers: (i) a device layer, which includes all devices with their corresponding wrappers, (ii) a communication layer, which enables the communication over an enterprise bus, and (iii) a processing and control layer, which includes event handlers, a database, and a connection to all higher layer services. The architecture was implemented in a real life office environment to demonstrate the effective handling of the system. The architecture proved sufficiently flexible, e.g., to efficiently integrate new devices by the use of wrappers, which abstract the device specific elements. Plug and Play capability was achieved by integrating wrappers into an OSGi environment and providing an open communication backbone with clear interfaces for services. All control and processing components are developed device-independent and can reuse services from existing components. For the definition of the control functionality, a rule-based system allowing modifications at run-time is provided.

The *Living Lab* with is flexible architecture provides a suitable framework for further investigation of smart grid capabilities, addressing different uses cases. The combination of production units together with storage capacity is used to demonstrate self-balancing of the power within the smart node, increasing the usage of own energy resources and reducing disturbances of the external grid. The inclusion of additional elements, like electric vehicles, will be addressed in an extension of the demonstrator. External data (e.g. price signals or weather forecasts) will also be integrated to optimize aspects like costs or reliability. User-interfaces for different devices like smart phones, tablet PCs or web front-ends are currently implemented to provide a suggestion mechanism to inform the user about the possibilities how energy can be saved. To ensure the reliability of the system – specifically when extending the rule system with more rules to provide better energy efficiency – a rule-checking mechanism is implemented to avoid the creation of unsound rule combinations. Furthermore, security concerns have to be improved as currently security was not the scope of the demonstrator. The architecture will be expanded to support security aspects as well, providing suitable roles and rights and access control mechanisms.

Finally, to better evaluate the suitability the architecture concerning its embedding in grid of prosumer nodes, future work will also focus on connecting the demonstrator with other demonstrators in Europe, as well as connecting the *Living Lab* to smart-grid simulation frameworks.

Acknowledgement

The authors would like to thank the EIT ICT Labs for the support to build up the demonstrator.

References

[1] B. Becker, F. Allerding, U. Reiner, M. Kahl, U. Richter, D. Pathmaperuma, H. Schmeck, and T. Leibfried, "Decentralized energy-management to control smart-home architectures," in *Architecture of Computing Systems - ARCS 2010*, ser. Lecture Notes in Computer Science, C. Mller-Schloer, W. Karl, and S. Yehia, Eds. Springer Berlin / Heidelberg, 2010, vol. 5974, pp. 150–161.

[2] EnOcean. (2012) EnOcean homepage. [Online]. Available: http://www.enocean-alliance.org/

[3] SMS-Guard. (2012) IPswitch-SG Anleitung. [Online]. Available: http://www.sms-guard.org/downloads/IPswitch-SG-Anleitung.pdf

[4] Siemens. (2012) SENTRON power monitoring device SENTRON PAC4200 - system manual. [Online]. Available: https://www.swe.siemens.com/portugal/web_nwa/pt/PortalInternet/QuemSomos/negocios/Industry/BT/LowVoltage/Documents/Sentron20PAC4200_brochura_EN.pdf

[5] IEC. (2012, February) IEC 61850-SER ed1.0 - Communication networks and systems in substations. [Online]. Available: http://webstore.iec.ch/Webstore/webstore.nsf/Artnum_PK/33549

[6] R. Ulrich, K. Mattias, T. Leibfried, A. Florian, and S. Hartmut, "Meregiomobil-labor - entwicklungsumgebung fr zuknftige smart-homes." in *VDE Kongress 2010 - E-Mobility*, 2010, p. N/A.

[7] F. Allerding and H. Schmeck, "Organic smart home: architecture for energy management in intelligent buildings," in *Proceedings of the 2011 workshop on Organic computing*, ser. OC '11. New York, NY, USA: ACM, 2011, pp. 67–76. [Online]. Available: http://doi.acm.org/10.1145/1998642.1998654

[8] K. Bao, F. Allerding, and H. Schmeck, "User behavior prediction for energy management in smart homes," in *Fuzzy Systems and Knowledge Discovery (FSKD), 2011 Eighth International Conference on*, vol. 2, july 2011, pp. 1335 – 1339.

[9] C.-S. Choi, J. Han, W.-K. Park, Y.-K. Jeong, and I.-W. Lee, "Proactive energy management system architecture interworking with smart grid," in *Consumer Electronics (ISCE), 2011 IEEE 15th International Symposium on*, june 2011, pp. 621 –624.

[10] G. Giaconia, G. Fiscelli, F. Bue, A. Di Stefano, D. La Cascia, and R. Miceli, "Integration of distributed on site control actions via combined photovoltaic and solar panels system," in *Clean Electrical Power, 2009 International Conference on*, june 2009, pp. 171 –177.

[11] K. Kok, S. Karnouskos, J. Ringelstein, A. Dimeas, A. Weidlich, C. Warmer, S. Drenkard, N. Hatziargyriou, and V. Lioliou, "Field-testing smart houses for a smart grid," in *21st International Conference and Exhibition on Electricity Distribution (CIRED 2011), Frankfurt, Germany*, 6-9 June 2011.

[12] J. Ma, S. Qin, B. Li, and T. Salsbury, "Economic model predictive control for building energy systems," in *Innovative Smart Grid Technologies (ISGT), 2011 IEEE PES*, jan. 2011, pp. 1 –6.

[13] H. Vogt and H. Weiss, "A client architecture for market-based grid integration of smart environments." in *PerCom Workshops'10*, 2010, pp. 642–647.

[14] I. Georgievski, V. Degeler, G. A. Pagani, T. A. Nguyen, A. Lazovik, and M. Aiello, "Optimizing offices for the smart grid," Distributed Systems Group, Institute for Mathematics and Computer Science, University of Groningen, Tech. Rep., November 2011.

[15] C.-L. Wu, C.-F. Liao, and L.-C. Fu, "Service-oriented smart-home architecture based on osgi and mobile-agent technology," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 37, no. 2, pp. 193 –205, march 2007.

[16] C. Liu, T. Reynolds, and H. Bao, "Research on SOA-based integrated strategy to enable the smart grid," in *Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2010 International Conference on*, oct. 2010, pp. 506 –509.

[17] T. Erl, *SOA Design Patterns*, 1st ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2009.

[18] D. S. Dirk Krafzig, Karl Banke, *Enterprise SOA: Service-Oriented Architecture Best Practices*. Prentice Hall PTR, 2004.

[19] Apache. (2012) Apache ActiveMQ homepage. [Online]. Available: http://activemq.apache.org/

[20] "JSR-000914 Java Message Service Specification Final Release 1.1," http://download.oracle.com/otndocs/jcp/7195-jms-1.1-fr-spec-oth-JSpec/, 04 202.

[21] Apache. (2012) Apache Camel homepage. [Online]. Available: http://camel.apache.org/

[22] R. S. Hall, K. Pauls, S. McCulloch, and D. Savage, *OSGi in Action : Creating Modular Applications in Java*. Manning, 2011.

[23] neo4j. (2012) neo4j homepage. [Online]. Available: http://neo4j.org/

[24] JBoss. (2012) JBoss Drools Homepage. [Online]. Available: http://www.jboss.org/drools

[25] D. Davis and M. P. Parashar, "Latency performance of soap implementations," in *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, ser. CCGRID '02. Washington, DC, USA: IEEE Computer Society, 2002, pp. 407–. [Online]. Available: http://dl.acm.org/citation.cfm?id=872748.873215

Appendix Hall 06

CFP last date 20 January 2025

International Journal of Computer Applications
A Publication of Foundation of Computer Science, Delaware
Scholarly peer reviewed publication

HOME ARCHIVES ABOUT US FOR AUTHORS

Home / Archives / Volume 44 / Number 13

Search

CALL FOR PAPER
February Edition

IJCA solicits high quality original research papers for the upcoming February edition of the journal. The last date of research paper submission is 20 January 2025

Submit your paper Know more

The week's pick



HYBRID ANN AND FIREWORKS ALGORITHM FOR REAL-TIME SOFTWARE RELIABILITY PREDICTION

Ivy Botchway Felix Larbi Aryeh Boniface Kayode Alese

Random Articles

AN ULTRA LOW POWERED MSP430 MICROCONTROLLER BASED CONTROL SYSTEM FOR A COMPOSTING PROCESS — February 2010

Investigation of Multi Band Microstrip Line Fed Antenna using DGS Technique for WLAN/WiMAX Applications — December 2013

Link Analysis to discover relevant documents using Information Retrieval — May 2019

A Survey on Infrastructure as Service Layer based Security Models for Cloud — May 2016

RESEACH ARTICLE

Decentralized Principles: New Modular Software Development Principles, A Robust Object Oriented Approach

by Gholamali Nejad Hajali Irani



International Journal of Computer Applications
Foundation of Computer Science (FCS), NY, USA
Volume 44 - Number 13
Year of Publication: 2012
Authors: Gholamali Nejad Hajali Irani
10.5120/6324-8671

ACMRef BibTeX EndNote

Gholamali Nejad Hajali Irani . Decentralized Principles: New Modular Software Development Principles, a Robust Object Oriented Approach. International Journal of Computer Applications. 44, 13 (April 2012), 26-33. DOI=10.5120/6324-8671

Abstract

Modularity as an object oriented principle helps to develop appropriate large-scale and complex software. But modularity has some deficits [14] such as modular decomposition etc. , which is not allowed widely using modularity in software development in current years. In this paper some principles have been provided for increase modularity of software systems and help for turn an existing system to a modular system. These principles distribute functionalities of each module to them and decrease dependency of modules. To obtain this aim, object oriented principles and heuristics has been analyzed then by considering a module as an object, new modular principles have been provided. In the reminder to evaluate new principles, a new modular architecture has been provided. The strength of new principles has been shown with two complete case studies. New principles can be used in any large-scale software architectures, modular architectures and any service oriented platforms.

References

1. A. J. Riel, Object-Oriented Design Heuristics, Addison Wesley, 1996.
2. The Content Management Comparision Tool, available at http://www. cmsmatrix. org
3. Recommended Practice for Architectural Description of Software Intensive Systems. Technical Report I EEE P1471-2000, IEEE Standards Department, The Architecture Working Group of the Software Engine ering Committee, 2000.
4. B. Boiko, Content Management Bible, 2nd Edition, Wiley Publishing, Inc. , Indianapolis, Indiana, 2005.
5. Drupal, Open Source CMS, available at http://Drupal. org/Project/Modules
6. A. Neto, H. Fernandes, D. Alves, D. F. Valc'arcel, B. B. Carvalho, J. Ferreira, et al, A standard data access layer for fusion devices R&D programs, Fusion Engineering and Design 82 (2007) 1315–1320.
7. S. Fiore, A. Negro, G. Aloisio, The data access layer in the GRelC system architecture, Future Generatio n Computer Systems 27 (2011) 334–340.
8. M. Butcher, G. Dunlap, M. Farina, L. Garfield, K. Rickard, J. Albin Wilkins, Drupal 7 Module Developmen t, Packt Publishing, 2010.
9. G. Manduchi, A. Luchetta, C. Taliercio, T. Fredian, J. Stillerman, Real-time data access layer for MDSplus, Fusion Engineering and Design 83 (2008) 312–316.
10. W. Mahnke, H. Steiert, Modularity in ORDBMSs – A new Challenge, Tagungsband 13. Workshop, Grun dlagen von Datenbanken, GI-FG 2. 5. 1, Magdeburg, Juni 2001.
11. O. Etzion, P. Niblett, Event Processing in Action, Manning Publications, USA, 2011.
12. R. Bourret, A. B. Coates, B. Harvey, G. K. Holman, M. Kay and et al, Advanced XML Applications from t he Experts at The XML Guild, Thomson Learning Inc, 2007.
13. D. Chappell, Enterprise Service Bus, O'Reilly Media, Inc, 2004.
14. B. Meyer, Object Oriented Software Construction, Second Edition, Prentice Hall, 1997.
15. J. X. Yuan, Liferay Portal 6 Enterprise Intranets, PACKT Publishing, 2010.
16. K. Pope, Zend Framework 1. 8 Web Application Development, PACKT Publishing, 2009.
17. R. Jay, SAP NetWeaver Portal Technology – The Complete Reference, McGraw Hill, 2008.

18. M. Shariff, V. Choudhary, A. Bhandari, P. Majmudar, Alfresco 3 Enterprise Content Management Implementation, PACKT Publishing, 2009.
19. R. S. Sanhu, Role hierarchies and constraints for lattice-based access controls, In Proceedings of the Fourth European Symposium on Research in Computer Security (ESORICS96,Rome, Italy, Sept. 25-27), E. Bertino, Ed. Springer-Verlag, New York, NY, 1996.
20. R. Castro-Rojo, D. R. López, The PAPI System: Point of Access to Providers of Information, Terena, 2001.
21. M. Steinemann, T. Spreng, A. Bachmayer, T. Braun, C. Graf, M. Guggisberg, Authentication and Authorization Infrastructure: Portal Architecture and Prototype Implementation, IAM-03-012, 2003.
22. G. Ahn, H. Hu, J. Jin. Security-Enhanced OSGi Service Environments, IEEE Transactions on Systems, Man and Cybernetics—Part C: Applications and Reviews, Vol. 39, No. 5, September 2009.
23. R. S. Hall, K. Pauls, S. McCulloch, D. Savage, OSGi in Action, Manning Publications, 2011.
24. C. Opincaru, G. Gheorghe, Service Oriented Security Architecture, 2008.
25. M. Fowler, D. Rice, M. Foemmel, E. Hieatt, R. Mee, R. Stafford. Patterns of Enterprise Application Architecture, Addison Wesley, 2002.
26. C. Nock. Data Access Patterns: Database Interactions in Object-Oriented Applications, Addison Wesley, 2003.
27. Java Persistence Layer Source Codes, available online at: http://Java-source. net/persistence.
28. C# Persistence Layer Source Codes, available online at: http://Csharp-source. net/persistence.
29. R. Barcia, G. Hambrick, K. Brown, R. Peterson, K. S. Bhogal, Persistence in the Enterprise: A Guide to Persistence Technologies, IBM Press, 2008.
30. F. Irmert, M. Daum, K. Meyer-Wegener, A New Approach to Modular Database Systems, SETMDM '08 Proceedings of the 2008 EDBT workshop on Software engineering for tailor-made data management ACM New York, NY, USA ©2008.
31. M. Mammarella, S. Hovsepian, E. Kohler, Modular Data Storage with Anvil, SOSP'09, October 11–14, 2009, Big Sky, Montana, USA.

Index Terms

Computer Science
└ Information Sciences

Keywords

Modular Software Architecture | Quality Attributes | Object Oriented Analysis And Design

Powered by PhDFocus™

This digital library is running on
PhDFocus™

IJCA is published by Foundation of Computer Science Inc.

© IJCA 2025

Terms of Service | Privacy Policy | Contact Us

Crossref

IJCA is a voting member of CrossRef. Each of the IJCA articles has its unique DOI reference.

Explore more details >

Decentralized Principles: New Modular Software Development Principles, a Robust Object Oriented Approach

GholamAli Nejad HajAli Irani
University of Bonab
Velayat Avenue, East Azerbaijan,
Bonab 5551761167, Iran

ABSTRACT
Modularity as an object oriented principle helps to develop appropriate large-scale and complex software. But modularity has some deficits [14] such as modular decomposition etc., which is not allowed widely using modularity in software development in current years. In this paper some principles have been provided for increase modularity of software systems and help for turn an existing system to a modular system. These principles distribute functionalities of each module to them and decrease dependency of modules.

To obtain this aim, object oriented principles and heuristics has been analyzed then by considering a module as an object, new modular principles have been provided. In the reminder to evaluate new principles, a new modular architecture has been provided. The strength of new principles has been shown with two complete case studies.

New principles can be used in any large-scale software architectures, modular architectures and any service oriented platforms.

Keywords
Modular Software Architecture, Quality Attributes, Object Oriented Analysis and Design.

1. INTRODUCTION
As increasing the scale and complexity of software systems, several types of approaches and architectures are proposed to overcome the complexity of there. Centralized approach is one of the proposed approaches. In this approach, the most common tasks of system have been extracted and assigned to the common part of system as named Core. Architectures consist of several components [3]. The main purpose of centralized approaches is to reduce the complexity of developing components. So components are developed quickly and many quality attributes of system is improved. In centralized approaches other components are highly depend on Core and if we change the Core, other components has changed as well. So extensibility, modifiability and flexibility of software are increased.

Modularity as an object oriented principle helps to have extensible, modifiable and flexible software. To reach perfect modular software we must have maximum amount of cohesion and minimum amount of coupling [14]. Therefore independent parts of software are excellent.

Based on [14], the biggest problem in modular development is decomposability problem, means that we can't always decompose each system to parts that are independent from each other [14]. So the big trade-off has been made between being modular and modular decomposition.

Most software has used centralized approaches. As increasing the scale and complexity of system, scale and complexity of Core is increasing as well. Then management of Core is turned to a big problem. On the other hand, while extending and modifying the Core, all modules might change. Therefore as increasing scale of Core, the Core can turn a GOD module [1], so we can't reach to modularity and written modules for a system can't be used in other systems.

For example, more than 1200 web portal and CMS is presented in web applications [2]. The CMS is composed of several modules [4]. For example Drupal is composed of more than 8700 modules [5]. So, each module has been written thousands times in general. These problems are due to lack of modularity.

In this paper, to reach a highest modularity between modules, a new modular principles based on decentralized approach have been provided and use object-oriented principles and heuristics, and distributes the Core complexity between other modules.

Steps of developing and evaluating new principles have been defined as following:

1. To examine incompetence of previous approaches and categorized requirement list.
2. To determine new principles of Modular Software Development.
3. To develop a new architecture and describe architecture modules, to prove the benefits of new principles.
4. To provide steps for developing new modules based on provided principles and describe how to use them.
5. To provide complete case studies based on new principles and develop new ideas.
6. To evaluate the new provided principles based on gathered requirement list.

2. PROBLEMS OF CENTRALIZED APPROACH
Being centralized in comparison with decentralized and modularized can cause numerous problems in development software which categorized as fallowing:

Req1: Modules have to use Core Functionalities (CF) – the common functionalities that have been centralized and implemented in Core. So, modules cohesion is increasing and dependency on Core is increasing as well, therefore modularity of system will be decreased.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0377

Req2: Developing small-scale modules need to follow the CF. Consequently; complexity of developing small-scale modules will be increased.

Req3: The implemented CF is not complete in general. Probably, developing large-scale modules is needed to use a new CF which is not supported by the Core.

Req4: Due to centralized approach and dependency of modules on Core, performing a Unit Test on modules is difficult and quality of testability is decreasing.

Req5: Due to variety of CF types and patterns, considering all of them in the Core cause to complexity of Core.

Req6: Because of centralized approach, integration of implemented modules into different systems with different Cores takes some efforts due to lack of standard interface. Therefore system integrity and modules portability decrease.

Req7: In centralized approach, the overall CP (so-called Big Picture) is apparent to all modules. So, encapsulation of CP is violated.

Table 1 presents the relationship obtained from the categorizing of above-mentioned requirements with software architecture quality attributes. In previous studies which are described in the reminder, none of above-mentioned problems are considered.

Table 1. Quality attributes affected by requirement list.

	E	M	Mo	I	Io	P	T	LEGEND
Req1			x					E: Extendibility;
Req2	x		x					M: Modifiability;
Req3	x	x						Mo: Modularity;
Req4							x	I: Integrity;
Req5	x				x			Io: Interoperability;
Req6			x	x		x		P: Portability;
Req7			x		x			T: Testability.

3. NEW MODULAR DEVELOPMENT PRINCIPLES

Based on table 1, theses problems are related to Modularity and in detail related to interoperability, portability, testability, extendibility and modifiability. In this section, to solving these problems, we used Object Oriented principles and heuristics in [1].

Based on [1], we can consider a module as an object then we can apply object oriented principles to obtain new modular development principles. List of used Object Oriented Heuristics from [1] are: Heuristic 2.1, 2.2, 2.4, 2.5, 2.6, 2.9, 2.10, 3.1, 3.2, 3.7, 3.8, 4.1, 4.2, 4.3, 4.4 and 5.3. By analyzing concepts of above-mentioned principles and heuristics and their relation with modularity concepts, we can provide new principles for developing modular systems which is shown in table 2.

Table 2. Mapping Object Oriented heuristics to Modular principles.

Heuristics	Code	Provided Principle
H2.1, H5.3	M1	Each module should hold and manage its own data by itself and Modules can't access each other data.
H2.9, H2.10	M2	Each module should perform all its functionalities by itself and Modules can't access each other functionalities.
H2.2, H2.3, H2.4, H2.5, H2.6, H4.1, H4.2, H4.3	M3	Minimize module relationships and interface. Independent modules are excellent.
H3.1, H3.2, H3.7, H3.8	M4	Beware of creation of God module [1]. (Minimize the Core).

4. EVALUATING PROVIDED PRINCIPLES

For evaluating and testing the provided principles which are shown in table 2, we proposed an architecture which was named MEDA. MEDA which is shown in figure 1 has been provided based on modular principles in table 2. In this architecture to reach maximum quantity of modularity and based on M2, each module has to perform all its functionality by itself and Core just run final instructions that have been sent by each module.

To reach maximum quantity of extendibility and modifiability, we need a dynamic structure and dynamic channel to communicate between modules. So, eXtensible Markup Language (XML) technologies [12] have been used to have dynamic structure and communication protocol between modules. To have dynamic channel, a combination of Event Driven Architecture [11] and a simple type of Bus Architecture [13] has been used. Event-Bus module in Core does these functionalities.

To support M1 and M4 from table 1, CAL (Content Access Layer) as a content layer has been placed in MEDA. CAL must execute instructions related to content access. Instruction is sent from modules to Event-Bus and then from Event-Bus to CAL. Contents can be Data, File or Web Services. In CAL, we need modular content access that modules can't access content of each other. For example in modular file access we must use operation system APIs that modules can't access files of each other. In modular data access we need assign a specific database username and password to each module that can't access table of each other.

Module Installer Layer must install and hold each modules and their profile. RTL (Run Time Layer) must do all runtime actions except content access instruction. Exception Manager must control and manage all exceptions that are sent from modules. AA Manager do authentication of whole system and based on M2, M4 authorization of system delivered to each module. Module Presentation must control presentation actions. Theme changing and language changing are done by Module Presentation. For example when a user changes the theme of CMS, theme changing as an event is sent to all modules by Module Presentation. Systems Decoration must manage template of CMS and locate modules presentation part in considered place.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0378

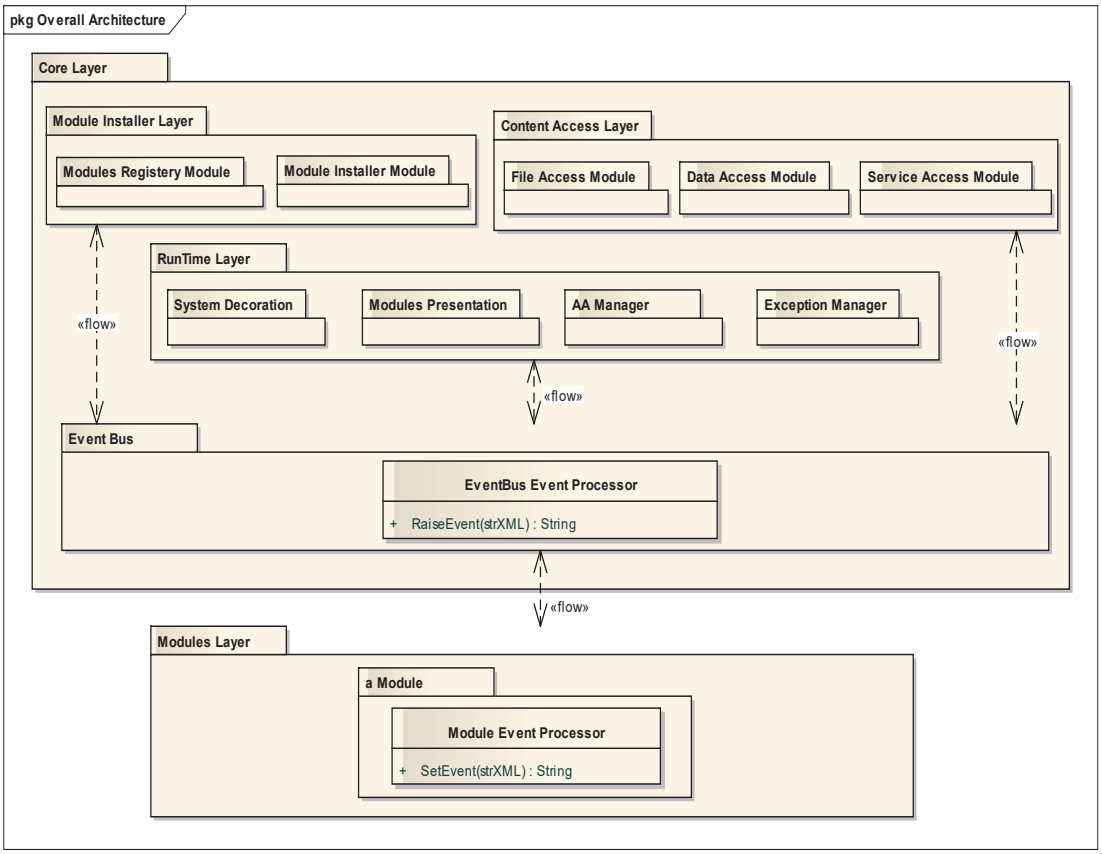


Figure 1: MEDA, provided architecture for evaluating new principles.

5. OVERALL SEQUENCE OF EVENT RAISING IN MEDA

In MEDA, each module and Core as a module, have to implement a class by the name of Event-Processor and use Event-Bus as a channel for interacting messages between Core and modules. In this architecture we put messages in the form of Events. All of modules and even the Core use RaiseEvent method from Event-Bus for sending events and Event-Bus uses SetEvent method from Event-Processor (implemented in each module) for sending delivered events to target modules.

Interaction between Core and modules are prepared by two types of XML files by the name of Document Type Definition 1 (DTD1) and DTD2. For example after a user logged in to system, Core will send a request (in the form of an event) to get User Access List from all the modules to represent User Control Panel. This action will perform by the use of an event like: +getAccessList(String Username):String;

DTD1 is a template for sent events. DTD1 usually contains event-type, event-name, input parameters names and values, return type and value, event-sender, event-receiver(s), etc. But while developing each module of MEDA, we must define standard DTD for module event list.

Each module for sending an event must put it in the form of DTD1 and invoke RaiseEvent method from Event-Bus.

Then Event-Bus analyze delivered event and in order to sending event to target modules, use the SetEvent method from Event-Processor class of each module. After that, all recipient modules can response to this event by returning value of SetEvent in the form of DTD2. DTD2 contains returned values of each module. Finally, Event-Bus put all of received DTD2s from each module in the <return> tag of DTD1 and passes the final DTD1 as return value of RaiseEvent.

6. STEPS FOR DEVELOPING MODULES

In this section, steps of developing modules based on provided principles are presented as following:

Step 1: Considering that the modular principles presented in table 2 is very general, we must derive detailed modular principles for desired module based on modular principles in table 2.

Step 2: To examine previous methods and approaches in modular systems for the module.

Step 3: To analyze and design of the module based on modular principles and decentralized approach. Use Case Model is necessary, because event list in following steps obtained from Use Case list.

Step 4: Considering M3, we must provide optimized event list of the module based on Use Case list. Event list must cover all use cases of module.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0379

Step 5: To standardize DTD and XML Schema for events of the module.

7. CASE STUDY 1: MODULAR AUTHENTICATION AND AUTHORIZATION

In this section steps of developing modular authentication and authorization (AA) has been provided. In step 1, detailed module principles for AA has provided in table 3.

Table 3. Derived principles for developing AA.

Base Principle	Obtained Principle
M4, M2	Each module has to perform its authorization by itself.
M4, M1	Each module has to hold and manage its authorization data.
M3	Standardize an AA interface between Core and Modules.

In step 2, we must collect and examine previous methods for AA. Previous studies of AA used a centralize approach, hence all AA data and its implementation is performed by Core [15]. However some studies distributed AA data into modules, Core is controlling and deciding about AA [16]. This centralized thinking has some inadequacy which will be discussed in the reminder.

For authenticating, some systems use a standard account management such as LDAP, SSO, NTLM, OpenID, OpenSSO and Site Minder etc [17], [18], [15], [16].

To perform authorization, a variety of resources of a system can be accessed by user in different levels. These resource types can be Application, Portlet, Locations and Files (Content-Model-Resources), Communications, Pages (or Forms), Use Cases (or Actions), Tables (or Database Entities), Objects (or Business Entities).

For authorizing, various approaches are provided. Role-Based Access Control (RBAC) is the common method for controlling user access to system resources and actions [19]. Some of the approached, for special uses, separated authentication from authorization [20]. These methods used federated user administration, therefore authentication performs in user's home system and authorization performs in service provider system [21]. To support extensibility and modifiability, AA methods used some new tools like Aspect Oriented Programming Languages [22]. Open Service Gateway initiative (OSGi) framework uses java standard called Java Authentication and Authorization Service which is a centralized approach [23].

Cristian and Gabriela showed that by distributing the security functions, a more flexible architecture can be designed that would lower the costs associated with implementation, administration and maintenance [24].

In step 3, we must perform analysis and design step for AA based on derived principles in table 3. Distilled use case model is shown in figure 2.

In step 4, we must provide optimized event list of AA based on its Use Case list. Optimized event list for AA is shown in table 4.

In step 5, we must standardize DTD and XML Schema for events of the module. Regard as this step outputs is large, an instance of DTD1 and DTD2 is shown in figure 3.

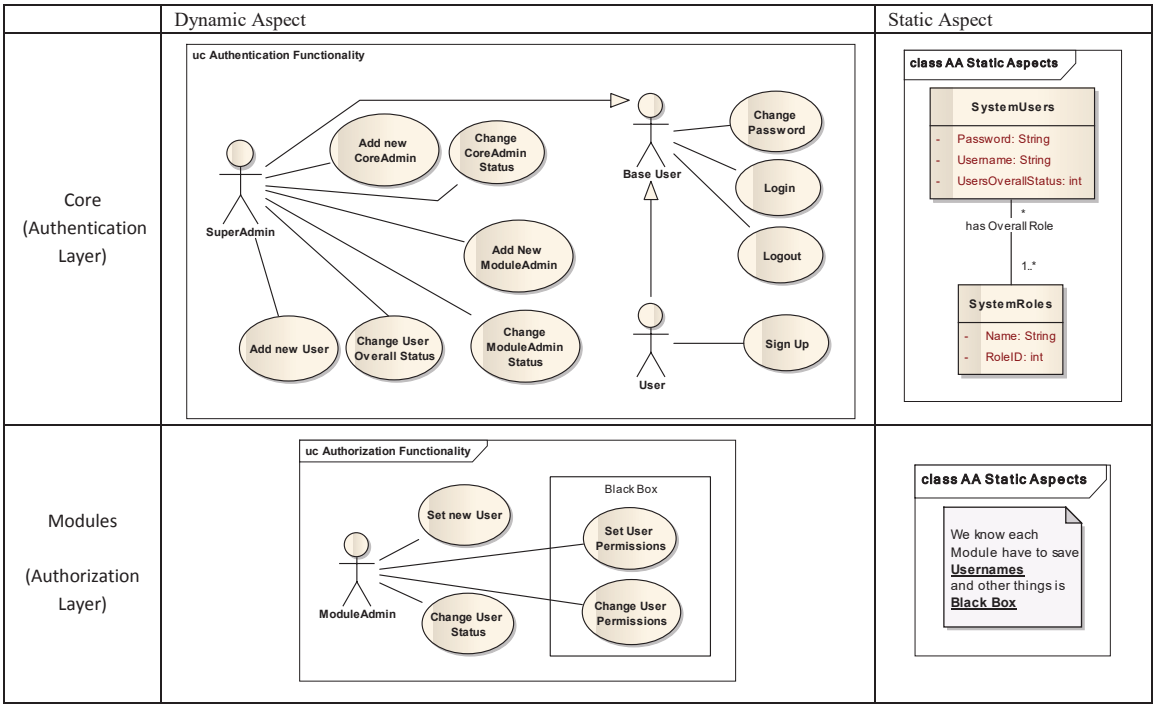


Figure 2: Distilled AA analysis and design.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0380

Table 4. Optimized event list for AA.

Use cases	Events	Description
New User, Sign Up	NewUser(String Username):void;	As soon as a new user registers in system, Core should inform all the modules, so the modules can grant default permissions to he/she.
New ModuleAdmin	NewModuleAdmin(String Username):void;	As soon as a new ModuleAdmin registers in system, Core should inform the target module.
Change User Overall Status	ChangeUserStatus(String Username, UserStatus status):void;	When Core changes overall status of a user, should inform all the modules.
Login	getAccessList(String Username):String;	For creating Control Panel for a user.
Other Use Cases	---	For other cases modules act independently.

```
<Event EventType="AAMNG"
EventName="getAccessList" EventID="123"
SenderName="Core" SenderID="379"
ReceiverNames="Core | a Module | *" ReceiverIDs="12 ,
32 | *" RaiseDateTime=" " Description=" ">

    <InputFields>
        <Field Name="UserName" Value="Jane"/>
    </InputFields>
    <Return Type="String">
        <!--return info must be in here in DTD2 format
-->
    </Return>
</Event>
```

```
<ReturnObjects ModuleName="News" ModuleID="123"
ReturnDateTime="" Name="News Access List"
Description="">
<Object>
        <Field Name="ID" Value="1"/>
        <Field Name="Title" Value="Add New News"/>
        <Field Name="URL"
Value="www.test.com/UI/News?1"/>
</Object>
<Object>
        <Field Name="ID" Value="2"/>
        <Field Name="Title" Value="Change News"/>
        <Field Name="URL"
Value="www.test.com/UI/News?2"/>
</Object>
</ReturnObjects>
```

Figure 3: (a) An instance of DTD1; (b) An instance of DTD2.

8. CASE STUDY 2: MODULAR DATA ACCESS ARCHITECTURE

In this section distilled steps of developing modular Data Access (DA) has been provided. In step 1, detailed module Software development principles for Data Access (DA) has provided in table 5.

Table 5. Derived principles for developing DA.

Code	Principle Code	Principle
P1	M4, M1	Each module has to manage its Data accessibility within itself.
P2	M4, M2	Each module has to perform its Data Management by itself.
P3	M3	Standardize a DA interface between Core and Modules and between modules.

In step 2, we must collect and examine previous methods for DA with decentralized approach.

DA methods in software architectures called Data Access Patterns (DAP). DAPs are architectures or patterns that manage the data access layer functionalities in Information Systems. Nowadays more than 50 DAPs have been developed [25]. Based on developed patterns from [25], [26] and etc, we categorized all existing DAPs in six groups which are shown in table 6.

There are not any modular data access patterns in provided patterns and they can't be used in our architecture.

Table 6. Provided category for exist DAPs.

Code	Description
DAP0	These methods don't use any data access layer and any parts of code directly connect to database. Due to high performance, these methods mostly are used in real time systems.
DAP1	These methods use one or more classes in data access layer to encapsulate database access details.
DAP2	These methods use DAP0. Also one entity class is created for each table in database and all CRUD (Create, Retrieve, Update and Delete) methods on this table are handled by its entity class.
DAP3	These methods are similar to DAP2 and entity classes are created for each table, but based on object oriented heuristics, all CRUD methods implemented in one class and other entity classes inherit from it.
DAP3M	These methods like DAP3, but metadata of all tables are stored in database as well. All entity classes inherit from Base Class. It create SQL command dynamically and manage them using stored metadata.
DAP4	In these methods unlike DAP3 and DAP2, entity classes for each table are not created. To gain excellent extendibility and modifiability (like DAP3M), metadata of all tables is stored in database and one or more classes perform all CRUD methods for all tables.

Persistence Frameworks (PF) are similar to DAPs and they can be used as a communication layer between applications and database. In [27] and [28] some of PFs are listed. In [29] other types of PFs and evaluation of them was provided as well. For example Hibernate is a PF for Java. PFs and DAPs

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0381

use similar approaches to communicate with database. All provided PFs are not modular as well.

Other Web Frameworks and Portals like SAP [17], Liferay [15], Zend [16], Alfresco [18] and Drupal [8] use a provided PFs or DAPs and none of them are modular. In [6], [7] and [9] new DAPs have been developed, but none of them are modular as well.

Another approach is Modular Database. In [10] modular databases have been suggested as a challenge and many aspects of it have been investigated. Afterwards, in [31] and [30] new architectures for modular database have been provided, but don't support all aspect of modularity and extendibility like event-driven. In next sections we will show that modular database can't be a modular data access layer and it is not a panacea for our problem.

In step 3, we must perform analysis and design step for modular DA based on derived principles in table 5. Distilled use case model is shown in figure 4.

In step 4, we must provide optimized event list of modular DA based on its Use Case list. Optimized event list for DA is shown in table 7.

Adding a new event will not affect DA in any way. For example, in order to gather Content Search, we can add an event such as: +getContentSearch(String ContentInfo):String; (from Core.DA to All Modules) into event list and then all modules can response to this event and Core after get all results from modules, complete final result and return it.

In step 5, we must standardize DTD and XML Schema for events of the module. Regard as this step outputs is large, an instance of DTD1 and DTD2 is shown in figure 5.

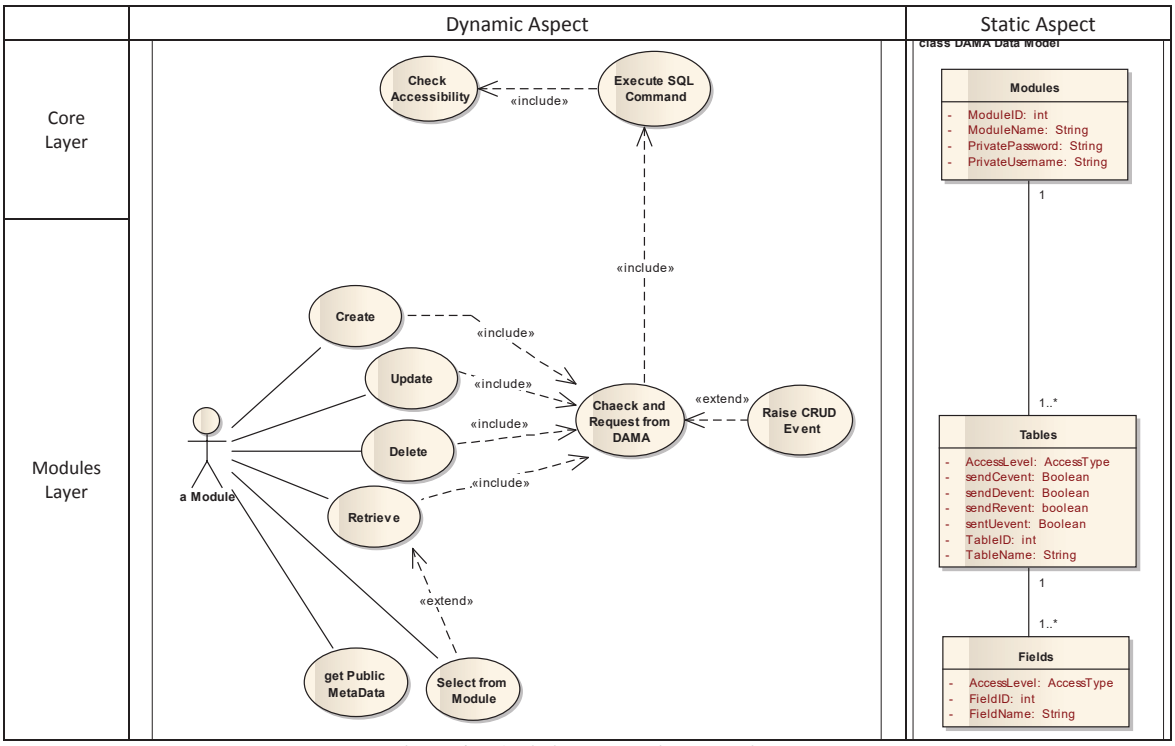


Figure 4. DA distilled analysis and design.

Table 7. Optimized Event List For DA.

Use cases	Events	from	to	Description
Create, Update, Delete	ExecuteSQL (String strSQL):String;	A Module	DA	When a module wants to perform a SQL command from its tables by DA, should use this event. As the type of Create, Update and Delete are different from Retrieve; they were categorized in different event.
Retrieve	ExecuteRetrieveSQL(String strSQL):String;	A Module	DA	When a module wants to Retrieve from its tables by DA, should use this event.
Select from a Module	RetrieveRequest(String strSQL):String;	A Module	Another Module	When a module wants to select from another modules tables, should use this event. This event performs after Get Public MetaDate event.
Get Public MetaData	getPublicMetaData(): String;	A Module	Another Module	Any modules must to response to this event that is sent from other modules and return its public access tables.
Raise CRUD Event	CRUDEvent(String CRUDInfo):void;	A Module	All Modules	Each module can raise a CRUD notice event after CRUD performs. With this event, modules can inform other modules.
Other Use Cases	---			For other cases, modules act independently or don't need to any event.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0382

```
<Event EventType="DAMA-Select"
EventName="SelectNews" EventID="123" SenderName="A
Module" SenderID="379" ReceiverNames=" B Module"
ReceiverIDs="12" RaiseDateTime=" " Description=" ">
        <InputFields>
                <Field Name="SQLCommand"
        Value="Select * from News where
        NewsNO<14"/>
        </InputFields>
        <Return Type="String">
                <!-- return info must be in here in
        DTD2 format -->
        </Return>
</Event>
```

```
<ReturnObjects ModuleName="B Module" ModuleID="123"
ReturnDateTime="" Name="NewsRow" Description="">
<Object>
        <Field Name="NewsNO" Value="10"/>
        <Field Name="NewsTitle" Value="news title1"/>
        <Field Name="NewsBody" Value="news body1"/>
</Object>
<Object>
        <Field Name="NewsID" Value="11"/>
        <Field Name="NewsTitle" Value="news title2"/>
        <Field Name="NewsBody" Value="news body2"/>
</Object>
</ReturnObjects>
```

Figure 5. (a) An instance of DTD1. (b) An instance of DTD2.

9. EVALUATION

In section 2, we categorized a requirement list as problems of centralized approaches. Decentralized principles capture all of these requirements which are shown in table 1. In fact, decentralized principles improve all quality attributes which are mentioned in table 8.

Table 8. Requirement list is captured by MEDA.

Description	Captured Requirements
Modules are independent in selecting their own patterns. They just have to consider Core's standard interface.	Req1, Req2, Req3
Since modules are not dependent to Core, unit test of each module can perform easily far from the Core.	Req4
Modules are free to choose their patterns and we don't need to collect all patterns in Core.	Req5
As establishing a new standard interface for Core and modules communication, integrity and portability of modules was increased and modularity of each module was increased to higher degree.	Req6, Req7

10. CONCLUSION

In this paper, new decentralized principles for modular systems have been provided. These principles distribute Core functionalities between modules based on robust object oriented thinking. So, dependencies between modules decrease saliently and existing systems turn to more modular systems. Therefore module development will take extra effort than before. Although it could be a disadvantage in comparison with centralized systems, this extra effort is worth benefiting of being decentralized.

These principles can be use in Service Oriented Platforms and any large-scale modular software. In addition, we can use these principles to any aspect of software. For example modular authentication and authorization and modular data access have provided as case studied. Modular exception handling, modular service access, modular file access and etc, can be investigate as future works of this paper.

11. REFERENCES

[1] A. J. Riel, Object-Oriented Design Heuristics, Addison Wesley, 1996.

[2] The Content Management Comparision Tool, available at http://www.cmsmatrix.org

[3] Recommended Practice for Architectural Description of Software Intensive Systems. Technical Report IEEE P1471-2000, IEEE Standards Department, The Architecture Working Group of the Software Engineering Committee, 2000.

[4] B. Boiko, Content Management Bible, 2nd Edition, Wiley Publishing, Inc., Indianapolis, Indiana, 2005.

[5] Drupal, Open Source CMS, available at http://Drupal.org/Project/Modules

[6] A. Neto, H. Fernandes, D. Alves, D.F. Valc´arcel, B.B. Carvalho, J. Ferreira, et al, A standard data access layer for fusion devices R&D programs, Fusion Engineering and Design 82 (2007) 1315–1320.

[7] S. Fiore, A. Negro, G. Aloisio, The data access layer in the GRelC system architecture, Future Generation Computer Systems 27 (2011) 334–340.

[8] M. Butcher, G. Dunlap, M. Farina, L. Garfield, K. Rickard, J. Albin Wilkins, Drupal 7 Module Development, Packt Publishing, 2010.

[9] G. Manduchi, A. Luchetta, C. Taliercio, T. Fredian, J. Stillerman, Real-time data access layer for MDSplus, Fusion Engineering and Design 83 (2008) 312–316.

[10] W. Mahnke, H. Steiert, Modularity in ORDBMSs – A new Challenge, Tagungsband 13. Workshop, Grundlagen von Datenbanken, GI-FG 2.5.1, Magdeburg, Juni 2001.

[11] O. Etzion, P. Niblett, Event Processing in Action, Manning Publications, USA, 2011.

[12] R. Bourret, A. B. Coates, B. Harvey, G. K. Holman, M. Kay and et al, Advanced XML Applications from the Experts at The XML Guild, Thomson Learning Inc, 2007.

[13] D. Chappell, Enterprise Service Bus, O'Reilly Media, Inc, 2004.

[14] B. Meyer, Object Oriented Software Construction, Second Edition, Prentice Hall, 1997.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0383

[15] J. X. Yuan, Liferay Portal 6 Enterprise Intranets, PACKT Publishing, 2010.

[16] K. Pope, Zend Framework 1.8 Web Application Development, PACKT Publishing, 2009.

[17] R. Jay, SAP NetWeaver Portal Technology – The Complete Reference, McGraw Hill, 2008.

[18] M. Shariff, V. Choudhary, A. Bhandari, P. Majmudar, Alfresco 3 Enterprise Content Management Implementation, PACKT Publishing, 2009.

[19] R. S. Sanhu, Role hierarchies and constraints for lattice-based access controls, In Proceedings of the Fourth European Symposium on Research in Computer Security (ESORICS96,Rome, Italy, Sept. 25-27), E. Bertino, Ed. Springer-Verlag, New York, NY, 1996.

[20] R. Castro-Rojo, D.R. López, The PAPI System: Point of Access to Providers of Information, Terena, 2001.

[21] M. Steinemann, T. Spreng, A. Bachmayer, T. Braun, C. Graf, M. Guggisberg, Authentication and Authorization Infrastructure: Portal Architecture and Prototype Implementation, IAM-03-012, 2003.

[22] G. Ahn, H. Hu, J. Jin. Security-Enhanced OSGi Service Environments, IEEE Transactions on Systems, Man and Cybernetics—Part C: Applications and Reviews, Vol. 39, No. 5, September 2009.

[23] R. S. Hall, K. Pauls, S. McCulloch, D. Savage, OSGi in Action, Manning Publications, 2011.

[24] C. Opincaru, G. Gheorghe, Service Oriented Security Architecture, 2008.

[25] M. Fowler, D. Rice, M. Foemmel, E. Hieatt, R. Mee, R. Stafford. Patterns of Enterprise Application Architecture, Addison Wesley, 2002.

[26] C. Nock. Data Access Patterns: Database Interactions in Object-Oriented Applications, Addison Wesley, 2003.

[27] Java Persistence Layer Source Codes, available online at: http://Java-source.net/persistence.

[28] C# Persistence Layer Source Codes, available online at: http://Csharp-source.net/persistence.

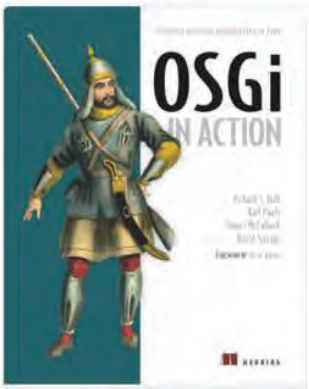
[29] R.Barcia, G.Hambrick, K.Brown, R.Peterson, K.S.Bhogal, Persistence in the Enterprise: A Guide to Persistence Technologies, IBM Press, 2008.

[30] F. Irmert, M. Daum, K. Meyer-Wegener, A New Approach to Modular Database Systems, SETMDM '08 Proceedings of the 2008 EDBT workshop on Software engineering for tailor-made data management ACM New York, NY, USA ©2008.

[31] M. Mammarella, S. Hovsepian, E. Kohler, Modular Data Storage with Anvil, SOSP'09, October 11–14, 2009, Big Sky, Montana, USA.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0384

Appendix Hall 07



look inside

resources

Source Code

Book Forum

you might also like



The book walks you through the JBoss 5 Application Server, from installation and configuration to... more



OSGi in Depth shows Java developers how to develop to the OSGi Service Platform Enterprise specification, ... more



Portlets in Action is a comprehensive guide to building portlet-driven applications in Java. It teaches portlet... more



Service Oriented Architecture, or SOA, has become embraced by many organizations as a means of improving... more

customers also reading



OSGi in Action

Creating Modular Applications in Java

Richard S. Hall, Karl Pauls, Stuart McCulloch, and David Savage

Foreword by Peter Kriens

April 2011 | ISBN 9781933988917 | 576 pages

printed in black & white

Java/JVM Enterprise Software Development

RECENTLY VIEWED

eBook
pdf, ePub, online

print
includes eBook

with subscription
free or 50% off

eBook
~~$25.99~~

add to cart

free with subscription

OSGi is a Java-based framework for creating applications as a set of interconnected modules. OSGi lets you install, start, stop, update, or uninstall modules at execution time without taking down your entire system. It's the backbone of the Eclipse plugin system, as well as many Java EE containers, such as GlassFish, Geronimo, and WebSphere.

OSGi in Action provides a clear introduction to OSGi concepts with examples that are relevant both for architects and developers. You'll start with the central ideas of OSGi: bundles, module lifecycles, and interaction among application components. With the core concepts well in hand, you'll explore numerous application scenarios and techniques. You'll learn how to migrate legacy systems to OSGi and how to test, debug, and manage applications.

"An impressive book."
Peter Kriens, OSGi Technical Director

RECENTLY VIEWED

about the technology

OSGi is a Java-based framework for creating applications as a set of interconnected modules. OSGi lets you install, start, stop, update, or uninstall modules at execution time without taking down your entire system. It's the backbone of the Eclipse plugin system, as well as many Java EE containers, such as GlassFish, Geronimo, and WebSphere.

"A lucid explanation of an intricate topic."
John S. Griffin, Overstock.com

"Easy to read...explains everything you need to know."
Jason Lee, Oracle

search inside this book

"Straight from the experts."
Erik Van Oosten, Travio

RECENTLY VIEWED

what's inside

- Core ideas of OSGi
- Vocabulary, tools, and strategies
- Applying OSGi

"Hit the ground running with this book."
David Dossot, Coauthor of Mule in Action

about the reader

This book assumes readers with a working knowledge of Java, but requires no previous exposure to OSGi.

about the authors

RECENTLY VIEWED

Richard S. Hall, **Karl Pauls**, **Stuart McCulloch**, and **David Savage** are all respected Java developers and committers on the Apache Felix OSGi implementation.

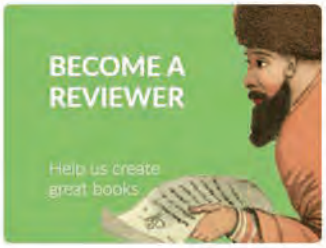


table of contents

You can see this entire book for free. Click anywhere in the table of contents to start reading

detailed TOC
FOREWORD
PREFACE
ACKNOWLEDGMENTS
ABOUT THIS BOOK
ABOUT THE AUTHORS

PART 1 INTRODUCING OSGI: MODULARITY, LIFECYCLE, AND SERVICES

1. OSGI REVEALED
2. MASTERING MODULARITY
3. LEARNING LIFECYCLE
4. STUDYING SERVICES
5. DELVING DEEPER INTO MODULARITY

PART 2 OSGI IN PRACTICE

6. MOVING TOWARD BUNDLES
7. TESTING APPLICATIONS
8. DEBUGGING APPLICATIONS
9. MANAGING BUNDLES
10. MANAGING APPLICATIONS

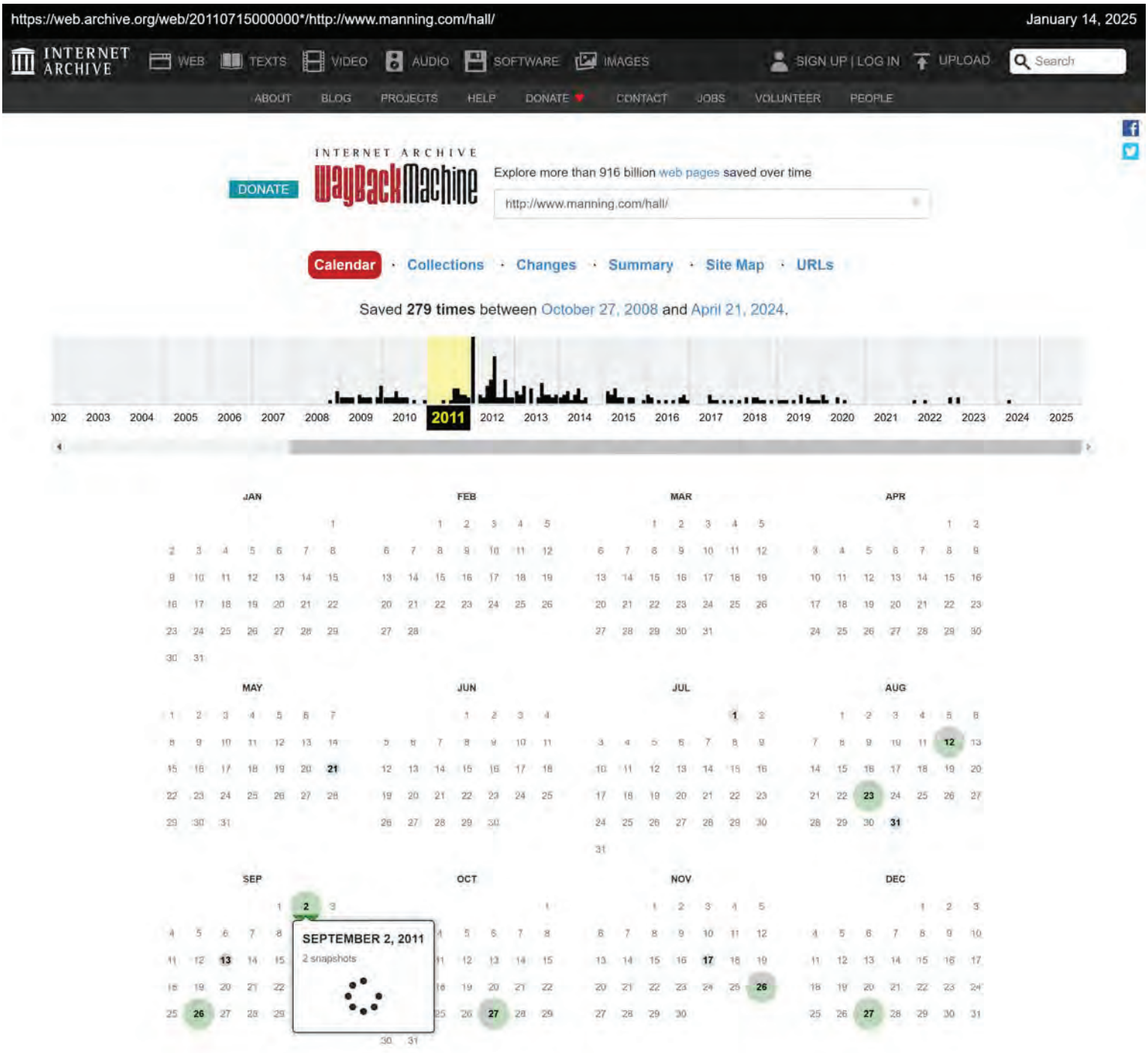
PART 3 ADVANCED TOPICS

11. COMPONENT MODELS AND FRAMEWORKS
12. ADVANCED COMPONENT FRAMEWORKS

RECENTLY VIEWED
RECENTLY VIEWED

Appendix Hall 08

https://web.archive.org/web/20110715000000*/http://www.manning.com/hall/ January 14, 2025



Note

This calendar view maps the number of times **http://www.manning.com/hall/** was crawled by the Wayback Machine, *not* how many times the site was actually updated. More info in the FAQ. Green indicates redirects (3xx).

FAQ | Contact Us | Terms of Service (Dec 31, 2014)

The Wayback Machine is an initiative of the Internet Archive, a 501(c)(3) non-profit building a digital library of Internet sites and other cultural artifacts in digital form. Other projects include Open Library & archive-it.org

Your use of the Wayback Machine is subject to the Internet Archive's Terms of Use.

https://web.archive.org/web/20110701134343/http://www.manning.com/hall/ January 14, 2025

INTERNET ARCHIVE Wayback Machine http://www.manning.com/hall/ Go MAY JUL 01 AUG 2010 2011 2012 279 captures 27 Oct 2008 - 21 Apr 2024 About this capture

Keep up with Manning for special offers and updates

Receive our email newsletter

Follow Manning on Twitter

Become a Manning Facebook fan

Catalog
Java
Microsoft & .NET
All by Title
All by Subject
Mobile Formats

Author Blogs
David Black
Tariq Ahmed
Jesus Garcia
Craig Walls
Joshua Blake
more...

Author Calendar
Manning Author Events
Manning Author Availability

About...
Manning
Contact Us
Ordering
Support
ebooks
Covers
Sandbox Forums
Distributors
Manning Early Access Program (MEAP)
Press
Academic
User Group Program
Jobs

Search



OSGi in Action

Creating Modular Applications in Java

Richard S. Hall, Karl Pauls, Stuart McCulloch, and David Savage

Foreword by Peter Kriens

April, 2011 | 576 pages
ISBN 1933988916

ADD TO CART **$49.99** Softbound print book - (includes free ebook)

ADD TO CART **$34.99** PDF ebook

RESOURCES

Look Inside
- Preface
- About this book
- Table of Contents
- Foreword
- Index

Resources
- Author Online
- OSGi in Action (Green Paper - PDF)

Downloads
- Source code (55.1 MB)
- Sample chapter 1
- Sample chapter 6

DESCRIPTION

OSGi is a Java-based framework for creating applications as a set of interconnected modules. OSGi lets you install, start, stop, update, or uninstall modules at execution time without taking down your entire system. It's the backbone of the Eclipse plugin system, as well as many Java EE containers, such as GlassFish, Geronimo, and WebSphere.

OSGi in Action provides a clear introduction to OSGi concepts with examples that are relevant both for architects and developers. You'll start with the central ideas of OSGi: bundles, module lifecycles, and interaction among application components. With the core concepts well in hand, you'll explore numerous application scenarios and techniques. You'll learn how to migrate legacy systems to OSGi and how to test, debug, and manage applications.

WHAT'S INSIDE

- Core ideas of OSGi
- Vocabulary, tools, and strategies
- Applying OSGi

This book assumes readers with a working knowledge of Java, but requires no previous exposure to OSGi.

ABOUT THE AUTHORS

Richard S. Hall, **Karl Pauls**, **Stuart McCulloch**, and **David Savage** are all respected Java developers and committers on the Apache Felix OSGi implementation.

WHAT REVIEWERS ARE SAYING

"Overall, this book is an excellent resource for Java developers of all levels of expertise in OSGi technology."
—Srini Penchikala, Security Architect

"I recommend this book to every serious Java developer which is involved or will be involved with modular architecture in general and OSGI in particular."
—Masoud Kalali, DZone Zone leader, author of GlassFish in Action

"An impressive book."
—From the foreword by Peter Kriens OSGi Technical Director

"A lucid explanation of an intricate topic."
—John S. Griffin, Overstock.com

"Easy to read ... explains everything you need to know."
—Jason Lee, Oracle

"Straight from the experts."
—Erik Van Oosten, JTeam

"Hit the ground running with this book."
—David Dossot Coauthor of *Mule in Action*

© 2003-2011 Manning Publications Co.



Creating modular applications in Java

OSGi IN ACTION

Richard S. Hall
Karl Pauls
Stuart McCulloch
David Savage

FOREWORD BY PETER KRIENS

SAMPLE CHAPTER

MANNING

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0391



OSGi in Action

by Richard S. Hall,
Karl Pauls,
Stuart McCulloch,
and David Savage

Chapter 1

Copyright 2011 Manning Publications

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0392

brief contents

PART 1 INTRODUCING OSGI: MODULARITY, LIFECYCLE, AND SERVICES 1

1 ■ OSGi revealed 3
2 ■ Mastering modularity 24
3 ■ Learning lifecycle 69
4 ■ Studying services 117
5 ■ Delving deeper into modularity 154

PART 2 OSGI IN PRACTICE 189

6 ■ Moving toward bundles 191
7 ■ Testing applications 230
8 ■ Debugging applications 258
9 ■ Managing bundles 292
10 ■ Managing applications 319

PART 3 ADVANCED TOPICS 343

11 ■ Component models and frameworks 345
12 ■ Advanced component frameworks 373

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0393

13 ■ Launching and embedding an OSGi framework 412
14 ■ Securing your applications 438
15 ■ Web applications and web services 477

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0394

Part 1

Introducing OSGi: modularity, lifecycle, and services

The OSGi framework defines a dynamic module system for Java. It gives you better control over the structure of your code, the ability to dynamically manage your code's lifecycle, and a loosely coupled approach for code collaboration. Even better, it's fully documented in a very elaborate specification. Unfortunately, the specification was written for people who are going to implement it rather than use it. In the first part of this book, we'll remedy this situation by effectively creating a user-oriented companion guide to the OSGi framework specification. We'll delve into its details by breaking it into three layers: module, lifecycle, and services. We'll explain what you need to understand from the specification to effectively use OSGi technology.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0395

1 OSGi revealed

This chapter covers

- Understanding Java's built-in support for modularity
- Introducing OSGi technology and how it improves Java modularity
- Positioning OSGi with respect to other technologies

The Java platform is an unqualified success story. It's used to develop applications for everything from small mobile devices to massive enterprise endeavors. This is a testament to its well-thought-out design and continued evolution. But this success has come in spite of the fact that Java doesn't have explicit support for building modular systems beyond ordinary object-oriented data encapsulation.

What does this mean to you? If Java is a success despite its lack of advanced modularization support, then you may wonder if that absence is a problem. Most well-managed projects have to build up a repertoire of project-specific techniques to compensate for the lack of modularization in Java. These include the following:

- Programming practices to capture logical structure
- Tricks with multiple class loaders
- Serialization between in-process components

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0397

But these techniques are inherently brittle and error prone because they aren't enforceable via any compile-time or execution-time checks. The end result has detrimental impacts on multiple stages of an application's lifecycle:

- *Development*—You're unable to clearly and explicitly partition development into independent pieces.
- *Deployment*—You're unable to easily analyze, understand, and resolve requirements imposed by the independently developed pieces composing a complete system.
- *Execution*—You're unable to manage and evolve the constituent pieces of a running system, nor minimize the impact of doing so.

It's possible to manage these issues in Java, and lots of projects do so using the custom techniques mentioned earlier, but it's much more difficult than it should be. We're tying ourselves in knots to work around the lack of a fundamental feature. If Java had explicit support for modularity, then you'd be freed from such issues and could concentrate on what you really want to do, which is developing the functionality of your application.

Welcome to the OSGi Service Platform. The OSGi Service Platform is an industry standard defined by the OSGi Alliance to specifically address the lack of support for modularity in the Java platform. As a continuation of its modularity support, it introduces a service-oriented programming model, referred to by some as *SOA in a VM*, to help you clearly separate interface from implementation. This chapter will give you an overview of the OSGi Service Platform and how it helps you create modular and manageable applications using an interface-based development model.

When we've finished this chapter, you'll understand what role OSGi technology plays among the arsenal of Java technologies and why Java and/or other Java-related technologies don't address the specific features provided by OSGi technology.

1.1 The what and why of OSGi

The $64,000 question is, "What is OSGi?" The simplest answer to this question is that it's a modularity layer for the Java platform. Of course, the next question that may spring to mind is, "What do you mean by *modularity*?" Here we use *modularity* more or less in the traditional computer-science sense, where the code of your software application is divided into logical parts representing separate concerns, as shown in figure 1.1. If your software is modular, you can simplify development and

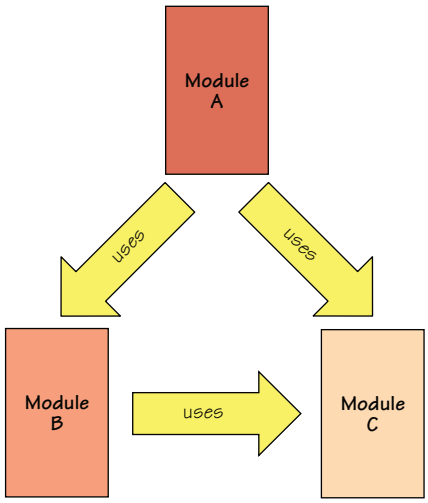


Figure 1.1 *Modularity* refers to the logical decomposition of a large system into smaller collaborating pieces.

improve maintainability by enforcing the logical module boundaries; we'll discuss more modularity details in chapter 2.

The notion of modularity isn't new. The concept became fashionable back in the 1970s. OSGi technology is cropping up all over the place—for example, as the runtime for the Eclipse IDE and the GlassFish application server. Why is it gaining popularity now? To better understand why OSGi is an increasingly important Java technology, it's worthwhile to understand some of Java's limitations with respect to creating modular applications. When you understand that, then you can see why OSGi technology is important and how it can help.

1.1.1 Java's modularity limitations

Java provides some aspects of modularity in the form of object orientation, but it was never intended to support coarse-grained modular programming. Although it's not fair to criticize Java for something it wasn't intended to address, the success of Java has resulted in difficulty for developers who ultimately have to deal with their need for better modularity support.

Java is promoted as a platform for building all sorts of applications for domains ranging from mobile phone to enterprise applications. Most of these endeavors require, or could at least benefit from, broader support for modularity. Let's look at some of Java's modularity limitations.

LOW-LEVEL CODE VISIBILITY CONTROL

Although Java provides a fair complement of access modifiers to control visibility (such as `public`, `protected`, `private`, and package private), these tend to address low-level object-oriented encapsulation and not logical system partitioning. Java has the notion of a *package*, which is typically used for partitioning code. For code to be visible from one Java package to another, the code must be declared `public` (or `protected` if using inheritance). Sometimes, the logical structure of your application calls for specific code to belong in different packages; but this means any dependencies among the packages must be exposed as `public`, which makes them accessible to everyone else, too. Often, this can expose implementation details, which makes future evolution more difficult because users may end up with dependencies on your nonpublic API.

To illustrate, let's consider a trivial "Hello, world!" application that provides a public interface in one package, a private implementation in another, and a main class in yet another.

Listing 1.1 Example of the limitations of Java's object-orientated encapsulation

Greeting.java

```
package org.foo.hello;

public interface Greeting {
  void sayHello();
}
```

(1) Simple interface

GreetingImpl.java

```
package org.foo.hello.impl;

import org.foo.hello.Greeting;
```

```
public class GreetingImpl implements Greeting {
  final String m_name;

  public GreetingImpl(String name) {
    m_name = name;
  }

  public void sayHello() {
    System.out.println("Hello, " + m_name + "!");
  }
}
```

2 Interface implementation

Main.java

```
package org.foo.hello.main;

import org.foo.hello.Greeting;
import org.foo.hello.impl.GreetingImpl;

public class Main {
  public static void main(String[] args) {
    Greeting greet = new GreetingImpl("Hello World");
    greet.sayHello();
  }
}
```

3 Main method

Listing 1.1's author may have intended a third party to only interact with the application via the `Greeting` interface 1. They may mention this in Javadoc, tutorials, blogs, or even email rants, but nothing stops a third party from constructing a new `GreetingImpl` using its public constructor 2 as is done at 3.

You may argue that the constructor shouldn't be public and that there is no need to split the application into multiple packages, which could well be true in this trivial example. But in real-world applications, class-level visibility when combined with packaging turns out to be a crude tool for ensuring API coherency. Because supposedly private implementation details can be accessed by third-party developers, you need to worry about changes to private implementation signatures as well as to public interfaces when making updates.

This problem stems from the fact that although Java packages appear to have a logical relationship via nested packages, they don't. A common misconception for people first learning Java is to assume that the parent-child package relationship bestows special visibility privileges on the involved packages. Two packages involved in a nested relationship are equivalent to two packages that aren't. Nested packages are largely useful for avoiding name clashes, but they provide only partial support for the logical code partitioning.

What this all means is that, in Java, you're regularly forced to decide between the following:

1. Impairing your application's logical structure by lumping unrelated classes into the same package to avoid exposing nonpublic APIs
2. Keeping your application's logical structure by using multiple packages at the expense of exposing nonpublic APIs so they can be accessed by classes in different packages

Neither choice is particularly palatable.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0400

ERROR-PRONE CLASS PATH CONCEPT

The Java platform also inhibits good modularity practices. The main culprit is the Java class path. Why does the class path pose problems for modularity? Largely due to all the issues it hides, such as code versions, dependencies, and consistency. Applications are generally composed of various versions of libraries and components. The class path pays no attention to code versions—it returns the first version it finds. Even if it did pay attention, there is no way to explicitly specify dependencies. The process of setting up your class path is largely trial and error; you just keep adding libraries until the VM stops complaining about missing classes.

Figure 1.2 shows the sort of "class path hell" often found when more than one JAR file provides a given set of classes. Even though each JAR file may have been compiled to work as a unit, when they're merged at execution time, the Java class path pays no attention to the logical partitioning of the components. This tends to lead to hard-to-predict errors, such as `NoSuchMethodError`, when a class from one JAR file interacts with an incompatible class version from another.

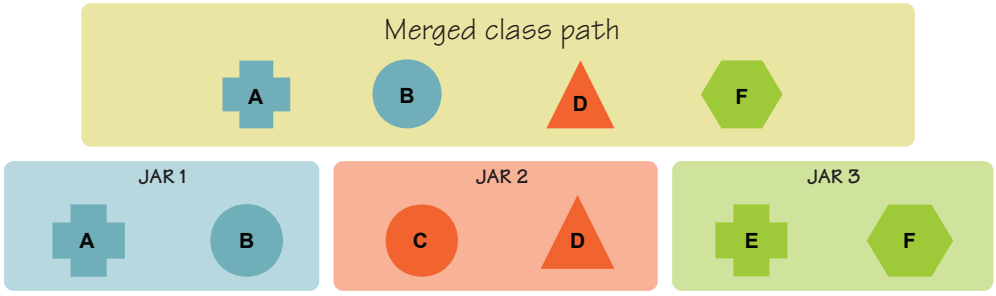


Figure 1.2 Multiple JARs containing overlapping classes and/or packages are merged based on their order of appearance in the class path, with no regard to logical coherency among archives.

In large applications created from independently developed components, it isn't uncommon to have dependencies on different versions of the same component, such as logging or XML parsing mechanisms. The class path forces you to choose one version in such situations, which may not always be possible. Worse, if you have multiple versions of the same package on the class path, either on purpose or accidentally, they're treated as split packages by Java and are implicitly merged based on order of appearance.

Overall, the class path approach lacks any form of consistency checking. You get whatever classes have been made available by the system administrator, which is likely only an approximation of what the developer expected.

LIMITED DEPLOYMENT AND MANAGEMENT SUPPORT

Java also lacks support when it comes to deploying and managing your application. There is no easy way in Java to deploy the proper transitive set of versioned code dependencies and execute your application. The same is true for evolving your application and its components after deployment.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0401

Consider the common requirement of wanting to support a dynamic plugin mechanism. The only way to achieve such a benign request is to use class loaders, which are low level and error prone. Class loaders were never intended to be a common tool for application developers, but many of today's systems require their use. A properly defined modularity layer for Java can deal with these issues by making the module concept explicit and raising the level of abstraction for code partitioning.

With this better understanding of Java's limitations when it comes to modularity, we can ponder whether OSGi is the right solution for your projects.

1.1.2 Can OSGi help you?

Nearly all but the simplest of applications can benefit from the modularity features OSGi provides, so if you're wondering if OSGi is something you should be interested in, the answer is most likely, "Yes!" Still not convinced? Here are some common scenarios you may have encountered where OSGi can be helpful:

- `ClassNotFoundException`s when starting your application because the class path wasn't correct. OSGi can help by ensuring that code dependencies are satisfied before allowing the code to execute.
- Execution-time errors from your application due to the wrong version of a dependent library on the class path. OSGi verifies that the set of dependencies are consistent with respect to required versions and other constraints.
- Type inconsistencies when sharing classes among modules: put more concretely, the dreaded appearance of `foo instanceof Foo == false`. With OSGi, you don't have to worry about the constraints implied by hierarchical classloading schemes.
- Packaging an application as logically independent JAR files and deploying only those pieces you need for a given installation. This pretty much describes the purpose of OSGi.
- Packaging an application as logically independent JAR files, declaring which code is accessible from each JAR file, and having this visibility enforced. OSGi enables a new level of code visibility for JAR files that allows you to specify what is and what isn't visible externally.
- Defining an extensibility mechanism for an application, like a plugin mechanism. OSGi modularity is particularly suited to providing a powerful extensibility mechanism, including support for execution-time dynamism.

As you can see, these scenarios cover a lot of use cases, but they're by no means exhaustive. The simple and non-intrusive nature of OSGi tends to make you discover more ways to apply it the more you use it. Having explored some of the limitations of the standard Java class path, we'll now properly introduce you to OSGi.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0402

1.2 An architectural overview of OSGi

The OSGi Service Platform is composed of two parts: the OSGi framework and OSGi standard services (depicted in figure 1.3). The framework is the runtime that implements and provides OSGi functionality. The standard services define reusable APIs for common tasks, such as Logging and Preferences.

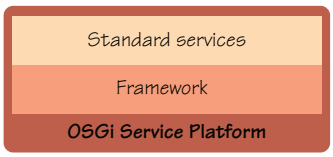


Figure 1.3 The OSGi Service Platform specification is divided into halves, one for the OSGi framework and one for standard services.

The OSGi specifications for the framework and standard services are managed by the OSGi Alliance (www.osgi.org/). The OSGi Alliance is an industry-backed nonprofit corporation founded in March 1999. The framework specification is now on its fourth major revision and is stable. Technology based on this specification is in use in a range of large-scale industry applications, including (but not limited to) automotive, mobile devices, desktop applications, and more recently enterprise application servers.

> **NOTE** Once upon a time, the letters *OSGi* were an acronym that stood for the Open Services Gateway Initiative. This acronym highlights the lineage of the technology but has fallen out of favor. After the third specification release, the OSGi Alliance officially dropped the acronym, and OSGi is now a trademark for the technology.

In the bulk of this book, we'll discuss the OSGi framework, its capabilities, and how to use these capabilities. Because there are so many standard services, we'll discuss only the most relevant and useful services, where appropriate. For any service we miss, you can get more information from the OSGi specifications. For now, we'll continue our overview of OSGi by introducing the broad features of the OSGi framework.

1.2.1 The OSGi framework

The OSGi framework plays a central role when you create OSGi-based applications, because it's the application's execution environment. The OSGi Alliance's framework specification defines the proper behavior of the framework, which gives you a well-defined API to program against. The specification also enables the creation of multiple implementations of the core framework to give you some freedom of choice; there are a handful of well-known open source projects, such as Apache Felix (http://felix.apache.org/), Eclipse Equinox (www.eclipse.org/equinox/), and Knopflerfish (www.knopflerfish.org/). This ultimately benefits you, because you aren't tied to a particular vendor and can program against the behavior defined in the specification. It's sort of like the reassuring feeling you get by knowing you can go into any McDonald's anywhere in the world and get the same meal!

OSGi technology is starting to pop up everywhere. You may not know it, but if you use an IDE to do your Java development, it's possible you already have experience with OSGi. The Equinox OSGi framework implementation is the underlying runtime for

the Eclipse IDE. Likewise, if you use the GlassFish v3 application server, you're also using OSGi, because the Apache Felix OSGi framework implementation is its runtime. The diversity of use cases attests to the value and flexibility provided by the OSGi framework through three conceptual layers defined in the OSGi specification (see figure 1.4):

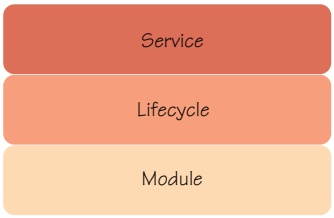


Figure 1.4
OSGi layered architecture

- *Module layer*—Concerned with packaging and sharing code
- *Lifecycle layer*—Concerned with providing execution-time module management and access to the underlying OSGi framework
- *Service layer*—Concerned with interaction and communication among modules, specifically the components contained in them

Like typical layered architectures, each layer is dependent on the layers beneath it. Therefore, it's possible for you to use lower OSGi layers without using upper ones, but not vice versa. The next three chapters discuss these layers in detail, but we'll give an overview of each here.

MODULE LAYER

The module layer defines the OSGi module concept, called a *bundle*, which is a JAR file with extra *metadata* (data about data). A bundle contains your class files and their related resources, as depicted in figure 1.5. Bundles typically aren't an entire application packaged into a single JAR file; rather, they're the logical modules that combine to form a given application. Bundles are more powerful than standard JAR files, because you can explicitly declare which contained packages are externally visible (that is, *exported packages*). In this sense, bundles extend the normal access modifiers (`public`, `private`, and `protected`) associated with the Java language.

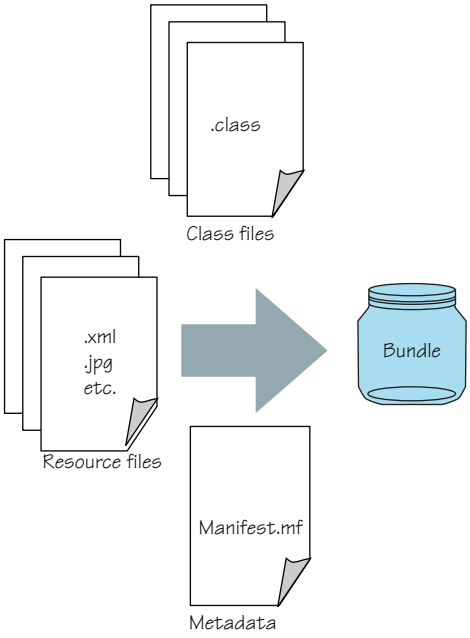


Figure 1.5 A bundle contains code, resources, and metadata.

Another important advantage of bundles over standard JAR files is the fact that you can explicitly declare on which external packages the bundles depend (that is, *imported packages*). The main benefit of explicitly declaring your bundles' exported and imported packages is that the OSGi framework can manage and verify their consistency automatically; this

process is called *bundle resolution* and involves matching exported packages to imported packages. Bundle resolution ensures consistency among bundles with respect to versions and other constraints, which we'll discuss in detail in chapter 2.

LIFECYCLE LAYER

The lifecycle layer defines how bundles are dynamically installed and managed in the OSGi framework. If you were building a house, the module layer would provide the foundation and structure, and the lifecycle layer would be the electrical wiring. It makes everything run.

The lifecycle layer serves two different purposes. External to your application, the lifecycle layer precisely defines the bundle lifecycle operations (install, update, start, stop, and uninstall). These lifecycle operations allow you to dynamically administer, manage, and evolve your application in a well-defined way. This means bundles can be safely added to and removed from the framework without restarting the application process.

Internal to your application, the lifecycle layer defines how your bundles gain access to their execution context, which provides them with a way to interact with the OSGi framework and the facilities it provides during execution. This overall approach to the lifecycle layer is powerful because it lets you create externally (and remotely) managed applications or completely self-managed applications (or any combination).

SERVICE LAYER

Finally, the service layer supports and promotes a flexible application programming model incorporating concepts popularized by service-oriented computing (although these concepts were part of the OSGi framework before service-oriented computing became popular). The main concepts revolve around the service-oriented publish, find, and bind interaction pattern: service providers publish their services into a service registry, while service clients search the registry to find available services to use (see figure 1.6). Nowadays, this service-oriented architecture (SOA) is largely associated with web services; but OSGi services are local to a single VM, which is why some people refer to it as *SOA in a VM*.

The OSGi service layer is intuitive, because it promotes an interface-based development approach, which is generally considered good practice. Specifically, it promotes the separation of interface and implementation. OSGi *services* are Java interfaces representing a conceptual contract between service providers and service clients. This makes the service layer lightweight, because service providers are just Java objects accessed via direct method invocation. Additionally, the service layer expands

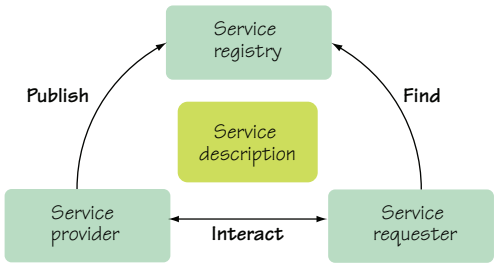


Figure 1.6 The service-oriented interaction pattern. Providers publish services into a registry where requesters can discover which services are available for use.

the bundle-based dynamism of the lifecycle layer with service-based dynamism—services can appear or disappear at any time. The result is a programming model eschewing the monolithic and brittle approaches of the past, in favor of being modular and flexible.

This sounds well and good, but you may still be wondering how these three layers fit together and how you go about using them to create an application on top of them. In the next couple of sections, we'll explore how these layers fit together using some small example programs.

1.2.2 Putting it all together

The OSGi framework is made up of layers, but how do you use these layers in application development? We'll make it clearer by outlining the general approach you'll use when creating an OSGi-based application:

1. Design your application by breaking it down into service interfaces (normal interface-based programming) and clients of those interfaces.
2. Implement your service provider and client components using your preferred tools and practices.
3. Package your service provider and client components into (usually) separate JAR files, augmenting each JAR file with the appropriate OSGi metadata.
4. Start the OSGi framework.
5. Install and start all your component JAR files from step 3.

If you're already following an interface-based approach, the OSGi approach will feel familiar. The main difference will be how you locate your interface implementations (that is, your services). Normally, you might instantiate implementations and pass around references to initialize clients. In the OSGi world, your services will publish themselves in the service registry, and your clients will look up available services in the registry. After your bundles are installed and started, your application will start and execute as normal, but with several advantages. Underneath, the OSGi framework provides more rigid modularity and consistency checking, and its dynamic nature opens up a world of possibilities.

Don't fret if you don't or can't use an interfaced-based approach for your development. The first two layers of the OSGi framework still provide a lot of functionality; in truth, the bulk of OSGi framework functionality lies in these first two layers, so keep reading. Enough talk: let's look at some code.

1.3 "Hello, world!" examples

Because OSGi functionality is divided over the three layers mentioned previously (modularity, lifecycle, and service), we'll show you three different "Hello, world!" examples that illustrate each of these layers.

1.3.1 Module layer example

The module layer isn't related to code creation as such; rather, it's related to the packaging of your code into bundles. You need to be aware of certain code-related issues

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0406

when developing, but by and large you prepare code for the module layer by adding packaging metadata to your project's generated JAR files. For example, suppose you want to share the following class.

Listing 1.2 Basic greeting implementation

```
package org.foo.hello;

public class Greeting {
  final String m_name;

  public Greeting(String name) {
    m_name = name;
  }

  public void sayHello() {
    System.out.println("Hello, " + m_name + "!");
  }
}
```

During the build process, you compile the source code and put the generated class file into a JAR file. To use the OSGi module layer, you must add some metadata into your JAR file's META-INF/MANIFEST.MF file, such as the following:

```
Bundle-ManifestVersion: 2
Bundle-Name: Greeting API
Bundle-SymbolicName: org.foo.hello
Bundle-Version: 1.0
Export-Package: org.foo.hello;version="1.0"
```

The first line indicates the OSGi metadata syntax version. Next is the human-readable name, which isn't strictly necessary. This is followed by the symbolic name and version bundle identifier. The last line shares packages with other bundles.

In this example, the bulk of the metadata is related to bundle identification. The important part is the `Export-Package` statement, because it extends the functionality of a typical JAR file with the ability for you to explicitly declare which packages contained in the JAR are visible to its users. In this example, only the contents of the `org.foo.hello` package are externally visible; if the example included other packages, they wouldn't be externally visible. This means that when you run your application, other modules won't be able to accidentally (or intentionally) depend on packages your module doesn't explicitly expose.

To use this shared code in another module, you again add metadata. This time, you use the `Import-Package` statement to explicitly declare which external packages are required by the code contained in the client JAR. The following snippet illustrates:

```
Bundle-ManifestVersion: 2
Bundle-Name: Greeting Client
Bundle-SymbolicName: org.foo.hello.client
Bundle-Version: 1.0
Import-Package: org.foo.hello;version="[1.0,2.0)"
```

In this case, the last line specifies a dependency on an external package.

To see this example in action, go in the chapter01/greeting-example/modularity/ directory in the book's companion code, and type `ant` to build it and `java -jar main.jar` to run it. Although the example is simple, it illustrates that creating OSGi bundles out of existing JAR files is a reasonably non-intrusive process. In addition, there are tools that can help you create your bundle metadata, which we'll discuss in appendix A; but in reality, no special tools are required to create a bundle other than what you normally use to create a JAR file. Chapter 2 will go into all the juicy details of OSGi modularity.

1.3.2 Lifecycle layer example

In the last subsection, you saw that it's possible to take advantage of OSGi functionality in a non-invasive way by adding metadata to your existing JAR files. Such a simple approach is sufficient for most reusable libraries, but sometimes you need or want to go further to meet specific requirements or to use additional OSGi features. The lifecycle layer moves you deeper into the OSGi world.

Perhaps you want to create a module that performs some initialization task, such as starting a background thread or initializing a driver; the lifecycle layer makes this possible. Bundles may declare a given class as an *activator*, which is the bundle's hook into its own lifecycle management. We'll discuss the full lifecycle of a bundle in chapter 3, but first let's look at a simple example to give you an idea of what we're talking about. The following listing extends the previous `Greeting` class to provide a singleton instance.

Listing 1.3 Extended greeting implementation

```
package org.foo.hello;

public class Greeting {
  static Greeting instance;
  final String m_name;

  Greeting(String name) {
    m_name = name;
  }

  public static Greeting get() {
    return instance;
  }

  public void sayHello() {
    System.out.println("Hello, " + m_name + "!");
  }
}
```

Singleton instance

Constructor now package private

Clients must use singleton

Listing 1.4 implements a bundle activator to initialize the `Greeting` class singleton when the bundle is started and clear it when it's stopped. The client can now use the preconfigured singleton instead of creating its own instance.

Listing 1.4 OSGi bundle activator for our greeting implementation

```
package org.foo.hello;

import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;

public class Activator implements BundleActivator {

  public void start(BundleContext ctx) {
    Greeting.instance = new Greeting("lifecycle");
  }

  public void stop(BundleContext ctx) {
    Greeting.instance = null;
  }
}
```

A bundle activator must implement a simple OSGi interface, which in this case is composed of the two methods `start()` and `stop()`. At execution time, the framework constructs an instance of this class and invokes the `start()` method when the bundle is started and the `stop()` method when the bundle is stopped. (What we mean by *starting* and *stopping* a bundle will become clearer in chapter 3.) Because the framework uses the same activator instance while the bundle is active, you can share member variables between the `start()` and `stop()` methods.

You may wonder what the single parameter of type `BundleContext` in the `start()` and `stop()` methods is all about. This is how the bundle gets access to the OSGi framework in which it's executing. From this context object, the module has access to all the OSGi functionality for modularity, lifecycle, and services. In short, it's a fairly important object for most bundles, but we'll defer a detailed introduction of it until later when we discuss the lifecycle layer. The important point to take away from this example is that bundles have a simple way to hook into their lifecycle and gain access to the underlying OSGi framework.

Of course, it isn't sufficient to just create this bundle activator implementation; you have to tell the framework about it. Luckily, this is simple. If you have an existing JAR file you're converting to be a module, you must add the activator implementation to the existing project so the class is included in the resulting JAR file. If you're creating a bundle from scratch, you need to compile the class and put the result in a JAR file. You must also tell the OSGi framework about the bundle activator by adding another piece of metadata to the JAR file manifest. For this section's example, you add the following metadata to the JAR manifest:

```
Bundle-Activator: org.foo.hello.Activator
Import-Package: org.osgi.framework
```

Notice that you also need to import the `org.osgi.framework` package, because the bundle activator has a dependency on it. To see this example in action, go to the chapter01/greeting-example/lifecycle/ directory in the companion code and type `ant` to build the example and `java -jar main.jar` to run it.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0409

We've now introduced how to create OSGi bundles out of existing JAR files using the module layer and how to make your bundles lifecycle aware so they can use framework functionality. The last example in this section demonstrates the service-oriented programming approach promoted by OSGi.

1.3.3 Service layer example

If you follow an interfaced-based approach in your development, the OSGi service approach will feel natural to you. To illustrate, consider the following `Greeting` interface:

```
package org.foo.hello;
public interface Greeting {
  void sayHello();
}
```

For any given implementation of the `Greeting` interface, when the `sayHello()` method is invoked, a greeting will be displayed. In general, a service represents a contract between a provider and prospective clients; the semantics of the contract are typically described in a separate, human-readable document, like a specification. The previous service interface represents the syntactic contract of all `Greeting` implementations. The notion of a contract is necessary so that clients can be assured of getting the functionality they expect when using a `Greeting` service.

The precise details of how any given `Greeting` implementation performs its task aren't known to the client. For example, one implementation may print its greeting textually, whereas another may display its greeting in a GUI dialog box. The following code depicts a simple text-based implementation.

Listing 1.5 Implementation of the `Greeting` interface

```
package org.foo.hello.impl;

import org.foo.hello.Greeting;

public class GreetingImpl implements Greeting {
  final String m_name;

  GreetingImpl(String name) {
    m_name = name;
  }

  public void sayHello() {
    System.out.println("Hello, " + m_name + "!");
  }
}
```

Your may be thinking that nothing in the service interface or listing 1.5 indicates that you're defining an OSGi service. You're correct. That's what makes the OSGi's service approach so natural if you're already following an interface-based approach; your code will largely stay the same. Your development will be a little different in two places: how you make a service instance available to the rest of your application, and how the rest of your application discovers the available service.

All service implementations are ultimately packaged into a bundle, and that bundle must be lifecycle aware in order to register the service. This means you need to create a bundle activator for the example service, as shown next.

Listing 1.6 OSGi bundle activator with service registration

```
package org.foo.hello.impl;

import org.foo.hello.Greeting;
import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;

public class Activator implements BundleActivator {

  public void start(BundleContext ctx) {
    ctx.registerService(Greeting.class.getName(),
        new GreetingImpl("service"), null);
  }

  public void stop(BundleContext ctx) {}
}
```

This time, in the `start()` method, instead of storing the `Greeting` implementation as a singleton, you use the provided bundle context to register it as a service in the service registry. The first parameter you need to provide is the interface name(s) that the service implements, followed by the actual service instance, and finally the service properties. In the `stop()` method, you could unregister the service implementation before stopping the bundle; but in practice, you don't need to do this. The OSGi framework automatically unregisters any registered services when a bundle stops.

You've seen how to register a service, but what about discovering a service? The following listing shows a simplistic client that doesn't handle missing services and that suffers from potential race conditions. We'll discuss a more robust way to access services in chapter 4.

Listing 1.7 OSGi bundle activator with service discovery

```
package org.foo.hello.client;

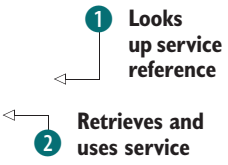
import org.foo.hello.Greeting;
import org.osgi.framework.*;

public class Client implements BundleActivator {

  public void start(BundleContext ctx) {
    ServiceReference ref =
        ctx.getServiceReference(Greeting.class.getName());

    ((Greeting) ctx.getService(ref)).sayHello();
  }

  public void stop(BundleContext ctx) {}
}
```



Notice that accessing a service in OSGi is a two-step process. First, an indirect reference is retrieved from the service registry (1). Second, this indirect reference is used to

access the service object instance ❷. The service reference can be safely stored in a member variable; but in general it isn't a good idea to hold on to references to service object instances, because services may be unregistered dynamically, resulting in stale references that prevent garbage collection of uninstalled bundles.

Both the service implementation and the client should be packaged into separate bundle JAR files. The metadata for each bundle declares its corresponding activator, but the service implementation exports the `org.foo.hello` package, whereas the client imports it. Note that the client bundle's metadata only needs to declare an import for the `Greeting` interface package—it has no direct dependency on the service implementation. This makes it easy to swap service implementations dynamically without restarting the client bundle. To see this example in action, go to the chapter01/greeting-example/service/ directory in the companion code and type `ant` to build the example and `java -jar main.jar` to run it.

Now that you've seen some examples, you can better understand how each layer of the OSGi framework builds on the previous one. Each layer gives you additional capabilities when building your application, but OSGi technology is flexible enough for you to adopt it according to your specific needs. If you only want better modularity in your project, use the module layer. If you want a way to initialize modules and interact with the module layer, use both the module and lifecycle layers. If you want a dynamic, interface-based development approach, use all three layers. The choice is yours.

1.3.4 Setting the stage

To help introduce the concepts of each layer in the OSGi framework in the next three chapters, we'll use a simple paint program; its user interface is shown in figure 1.7.

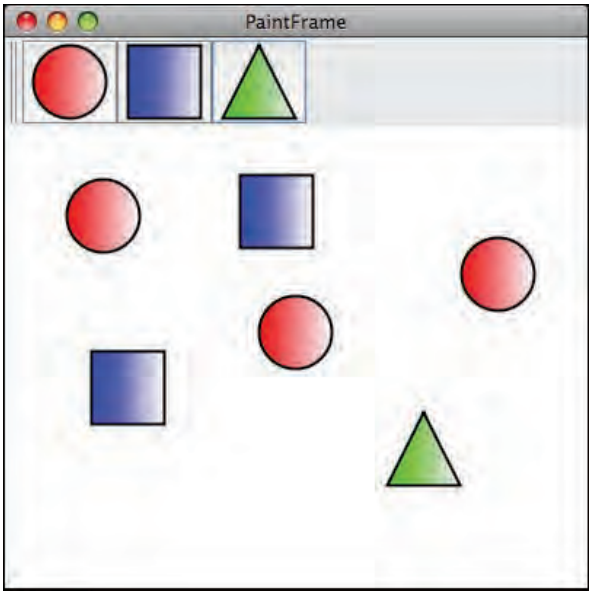


Figure 1.7 Simple paint program user interface

The paint program isn't intended to be independently useful; rather, it's used to demonstrate common issues and best practices.

From a functionality perspective, the paint program only allows the user to paint various shapes, such as circles, squares, and triangles. The shapes are painted in predefined colors. Available shapes are displayed as buttons in the main window's toolbar. To draw a shape, the user selects it in the toolbar and then clicks anywhere in the canvas to draw it. The same shape can be drawn repeatedly by clicking in the canvas numerous times. The user can drag drawn shapes to reposition them. This sounds simple enough. The real value of using a visual program for demonstrating these concepts will become evident when we start introducing execution-time dynamism.

We've finished our overview of the OSGi framework and are ready to delve into the details; but before we do, we'll put OSGi in context by discussing similar or related technologies. Although no Java technology fills the exact same niche as OSGi, several tread similar ground, and it's worth understanding their relevance before moving forward.

1.4 Putting OSGi in context

OSGi is often mentioned in the same breath with many other technologies, but it's in a fairly unique position in the Java world. Over the years, no single technology has addressed OSGi's exact problem space, but there have been overlaps, complements, and offshoots. Although it isn't possible to cover how OSGi relates to every conceivable technology, we'll address some of the most relevant in roughly chronological order. After reading this section, you should have a good idea whether OSGi replaces your familiar technologies or is complementary to them.

1.4.1 Java Enterprise Edition

Java Enterprise Edition (Java EE, formerly J2EE) has roots dating back to 1997. Java EE and OSGi began targeting opposite ends of the computing spectrum (the enterprise vs. embedded markets, respectively). Only within the last couple of years has OSGi technology begun to take root in the enterprise space.

In total, the Java EE API stack isn't related to OSGi. The Enterprise JavaBeans (EJB) specification is probably the closest comparable technology from the Java EE space, because it defines a component model and packaging format. But its component model focuses on providing a standard way to implement enterprise applications that must regularly handle issues of persistence, transactions, and security. The EJB deployment descriptors and packaging formats are relatively simplistic and don't address the full component lifecycle, nor do they support clean modularity concepts.

OSGi is also used in the Java EE domain to provide a more sophisticated module layer beneath these existing technologies. Because the two ignored each other for so long, there are some challenges in moving existing Java EE concepts to OSGi, largely due to different assumptions about how class loading is performed. Still, progress is being made, and today OSGi plays a role in all major application servers, such as IBM's WebSphere, Red Hat's JBoss, Oracle's GlassFish, ObjectWeb's JOnAS, and Apache's Geronimo.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0413

1.4.2 Jini

An often-overlooked Java technology is Jini, which is definitely a conceptual sibling of OSGi. Jini targets OSGi's original problem space of networked environments with a variety of connected devices.

Sun began developing Jini in 1998. The goal of Jini is to make it possible to administer a networked environment as a flexible, dynamic group of services. Jini introduces the concepts of service providers, service consumers, and a service lookup registry. All of this sounds completely isomorphic to OSGi; where Jini differs is its focus on distributed systems. Consumers access clients through some form of proxy using a remote procedure call mechanism, such as Remote Method Invocation (RMI). The service-lookup registry is also a remotely accessible, federated service. The Jini model assumes remote access across multiple VM processes, whereas OSGi assumes everything occurs in a single VM process. But in stark contrast to OSGi, Jini doesn't define any modularity mechanisms and relies on the execution-time code-loading features of RMI. The open source project Newton is an example of combining OSGi and Jini technologies in a single framework.

1.4.3 NetBeans

NetBeans, an IDE and application platform for Java, has a long history of having a modular design. Sun purchased NetBeans in 1999 and has continued to evolve it.

The NetBeans platform has a lot in common with OSGi. It defines a fairly sophisticated module layer and also promotes interface-based programming using a lookup pattern that is similar to the OSGi service registry. Whereas OSGi focused on embedded devices and dynamism, the NetBeans platform was originally an implementation layer for the IDE. Eventually, the platform was promoted as a separate tool in its own right, but it focused on being a complete GUI application platform with abstractions for file systems, windowing systems, and much more. NetBeans has never been seen as comparable to OSGi, even though it is; perhaps OSGi's more narrow focus is an asset in this case.

1.4.4 Java Management Extensions

Java Management Extensions (JMX), released in 2000 through the Java Community Process (JCP) as JSR 3, was compared to OSGi in the early days. JMX is a technology for remotely managing and monitoring applications, system objects, and devices; it defines a server and a component model for this purpose.

JMX isn't comparable to OSGi; it's complementary, because it can be used to manage and monitor an OSGi framework and its bundles and services. Why did the comparisons arise in the first place? There are probably three reasons: the JMX component model was sufficiently generic that it was possible to use it for building applications; the specification defined a mechanism for dynamically loading code into its server; and certain early adopters pushed JMX in this direction. One major perpetrator was JBoss, which adopted and extended JMX for use as a module layer in its

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0414

application server (since eliminated in JBoss 5). Nowadays, JMX isn't (and shouldn't be) confused with a module system.

1.4.5 Lightweight containers

Around 2003, lightweight or inversion of control (IoC) containers started to appear, such as PicoContainer, Spring, and Apache Avalon. The main idea behind this crop of IoC containers was to simplify component configuration and assembly by eliminating the use of concrete types in favor of interfaces. This was combined with dependency injection techniques, where components depend on interface types and implementations of the interfaces are injected into the component instance. OSGi services promote a similar interface-based approach but employ a service-locator pattern to break a component's dependency on component implementations, similar to Apache Avalon.

At the same time, the Service Binder project was creating a dependency injection framework for OSGi components. It's fairly easy to see why the comparisons arose. Regardless, OSGi's use of interface-based services and the service-locator pattern long predated this trend, and none of these technologies offer a sophisticated dynamic module layer like OSGi. There is now significant movement from IoC vendors to port their infrastructures to the OSGi framework, such as the work by VMware (formerly SpringSource) on the OSGi Blueprint specification (discussed in chapter 12).

1.4.6 Java Business Integration

Java Business Integration (JBI) was developed in the JCP and released in 2005. Its goal was to create a standard SOA platform for creating enterprise application integration (EAI) and business-to-business (B2B) integration solutions.

In JBI, plugin components provide and consume services after they're plugged in to the JBI framework. Components don't directly interact with services, as in OSGi; instead, they communicate indirectly using normalized Web Services Description Language (WSDL)-based messages.

JBI uses a JMX-based approach to manage component installation and lifecycle and defines a packaging format for its components. Due to the inherent similarities to OSGi's architecture, it was easy to think JBI was competing for a similar role. On the contrary, its fairly simplistic modularity mechanisms mainly addressed basic component integration into the framework. It made more sense for JBI to use OSGi's more sophisticated modularity, which is ultimately what happened in Project Fuji from Sun and ServiceMix from Apache.

1.4.7 JSR 277

In 2005, Sun announced JSR 277 ("Java Module System") to define a module system for Java. JSR 277 was intended to define a module framework, packaging format, and repository system for the Java platform. From the perspective of the OSGi Alliance, this was a major case of reinventing the wheel, because the effort was starting from scratch rather than building on the experience gained from OSGi.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0415

In 2006, many OSGi supporters pushed for the introduction of JSR 291 (titled "Dynamic Component Support for Java"), which was an effort to bring OSGi technology properly into JCP standardization. The goal was twofold: to create a bridge between the two communities and to ensure OSGi technology integration was considered by JSR 277. The completion of JSR 291 was fairly quick because it started from the OSGi R4 specification and resulted in the R4.1 specification release. During this period, OSGi technology continued to gain momentum. JSR 277 continued to make slow progress through 2008 until it was put on hold indefinitely.

1.4.8 JSR 294

During this time in 2006, JSR 294 (titled "Improved Modularity Support in the Java Programming Language") was introduced as an offshoot of JSR 277. Its goal was to focus on necessary language changes for modularity. The original idea was to introduce the notion of a *superpackage* into the Java language—a package of packages.

The specification of superpackages got bogged down in details until it was scrapped in favor of adding a module-access modifier keyword to the language. This simplification ultimately led to JSR 294 being dropped and merged back into JSR 277 in 2007. But when it became apparent in 2008 that JSR 277 would be put on hold, JSR 294 was pulled back out to address a module-level access modifier.

With JSR 277 on hold, Sun introduced an internal project, called *Project Jigsaw*, to modularize the JDK. The details of Jigsaw are still evolving after the acquisition of Sun by Oracle.

1.4.9 Service Component Architecture

Service Component Architecture (SCA) began as an industry collaboration in 2004 and ultimately resulted in final specifications in 2007. SCA defines a service-oriented component model similar to OSGi's, where components provide and require services. Its component model is more advanced because it defines *composite components* (components made of other components) for a fully recursive component model.

SCA is intended to be a component model for declaratively composing components implemented using various technologies (such as Java, Business Process Execution Language [BPEL], EJB, and C++) and integrated using various bindings (such as SOAP/HTTP, Java Message Service [JMS], Java EE Connector Architecture [JCA], and Internet Inter-Orb Protocol [IIOP]). SCA does define a standard packaging format, but it doesn't define a sophisticated module layer like OSGi provides. The SCA specification leaves open the possibility of other packaging formats, which makes it possible to use OSGi as a packaging and module layer for Java-based SCA implementations; Apache Tuscany and Newton are examples of an SCA implementation that use OSGi. In addition, bundles could be used to implement SCA component types, and SCA could be used as a mechanism to provide remote access to OSGi services.

1.4.10 .NET

Although Microsoft's .NET (released in 2002) isn't a Java technology, it deserves mention because it was largely inspired by Java and did improve on it in ways that are similar

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0416

to how OSGi improves Java. Microsoft not only learned from Java's example but also learned from the company's own history of dealing with DLL hell. As a result, .NET includes the notion of an *assembly*, which has modularity aspects similar to an OSGi bundle. All .NET code is packaged into an assembly, which takes the form of a DLL or EXE file. Assemblies provide an encapsulation mechanism for the code contained inside of them; an access modifier, called `internal`, is used to indicate visibility within an assembly but not external to it. Assemblies also contain metadata describing dependencies on other assemblies, but the overall model isn't as flexible as OSGi's. Because dependencies are on specific assembly versions, the OSGi notion of provider substitutability isn't attainable.

At execution time, assemblies are loaded into application domains and can only be unloaded by unloading the entire application domain. This makes the highly dynamic and lightweight nature of OSGi hard to achieve, because multiple assemblies loaded into the same application domain must be unloaded at the same time. It's possible to load assemblies into separate domains; but then communication across domains must use interprocess communication to collaborate, and type sharing is greatly complicated. There have been research efforts to create OSGi-like environments for the .NET platform, but the innate differences between the .NET and Java platforms results in the two not having much in common. Regardless, .NET deserves credit for improving on standard Java in this area.

1.5 Summary

In this chapter, we've laid the foundation for everything we'll cover in the rest of the book. What you've learned includes the following:

- The Java platform is great for developing applications, but its support for modularity is largely limited to fine-grained object-oriented mechanisms, rather than more coarse-grained modularity features needed for project management.
- The OSGi Service Platform, through the OSGi framework, addresses the modularity shortcomings of Java to create a powerful and flexible solution.
- The declarative, metadata-based approach employed by OSGi provides a non-invasive way to take advantage of its sophisticated modularity capabilities by modifying how projects are packaged with few, if any, changes to the code.
- The OSGi framework defines a controlled, dynamic module lifecycle to simplify management.
- Following good design principles, OSGi promotes an interface-based programming approach to separate interfaces from implementations.

With this high-level understanding of Java's limitations and OSGi's capabilities, we can start our adventure by diving into the details of the module layer in chapter 2. This is the foundation of everything else in the OSGi world.

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0417

JAVA

OSGi IN ACTION

Hall • Pauls • McCulloch • Savage

OSGi is a Java-based framework for creating applications as a set of interconnected modules. OSGi lets you install, start, stop, update, or uninstall modules at execution time without taking down your entire system. It's the backbone of the Eclipse plugin system, as well as many Java EE containers, such as GlassFish, Geronimo, and WebSphere.

OSGi in Action provides a clear introduction to OSGi concepts with examples that are relevant both for architects and developers. You'll start with the central ideas of OSGi: bundles, module lifecycles, and interaction among application components. With the core concepts well in hand, you'll explore numerous application scenarios and techniques. You'll learn how to migrate legacy systems to OSGi and how to test, debug, and manage applications.

What's Inside

- Core ideas of OSGi
- Vocabulary, tools, and strategies
- Applying OSGi

This book assumes readers with a working knowledge of Java, but requires no previous exposure to OSGi.

Richard S. Hall, **Karl Pauls**, **Stuart McCulloch**, and **David Savage** are all respected Java developers and committers on the Apache Felix OSGi implementation.

For access to the book's forum and a free ebook for owners of this book, go to manning.com/OSGiinAction

Free eBook
SEE INSERT

"An impressive book."
—From the foreword by Peter Kriens
OSGi Technical Director

"A lucid explanation of an intricate topic."
—John S. Griffin, Overstock.com

"Easy to read ... explains everything you need to know."
—Jason Lee, Oracle

"Straight from the experts."
—Erik Van Oosten, JTeam

"Hit the ground running with this book."
—David Dossot
Coauthor of *Mule in Action*

ISBN-13: 978-1-933988-91-7
ISBN-10: 1-933988-91-6
54999
9 781933 988917

MANNING $49.99 / Can $61.99 [INCLUDING eBOOK]

Exhibit 1014
Microsoft Corporation v. Edge Networking Systems, LLC
0418