

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Patent of: Sanjay Dahiya
U.S. Patent No.: 7,640,332 Attorney Docket No.: 39898-0051IP1
Issue Date: December 29, 2009
Appl. Serial No.: 12/003,182
Filing Date: December 20, 2007
Title: SYSTEM AND METHOD FOR HOT
DEPLOYMENT/REDEPLOYMENT IN GRID
COMPUTING ENVIRONMENT

Mail Stop Patent Board

Patent Trial and Appeal Board
U.S. Patent and Trademark Office
P.O. Box 1450
Alexandria, VA 22313-1450

**PETITION FOR *INTER PARTES* REVIEW OF UNITED STATES PATENT
NO. 7,640,332 PURSUANT TO 35 U.S.C. §§ 311–319, 37 C.F.R. § 42**

TABLE OF CONTENTS

LIST OF EXHIBITS	iii
LISTING OF CLAIMS	v
CONVENTIONS OF THE PETITION	xi
I. REQUIREMENTS FOR IPR—37 C.F.R. §42.104.....	1
A. Grounds for Standing—37 C.F.R. §42.104(a).....	1
B. Challenge and Relief Requested—37 C.F.R. §42.104(b)	1
II. THE '332 PATENT	2
A. Pertinent Disclosure	2
B. Prosecution History.....	4
III. LEVEL OF ORDINARY SKILL	4
IV. CLAIM CONSTRUCTION—37 C.F.R. §42.104(b)(3)	4
V. THE CHALLENGED CLAIMS ARE UNPATENTABLE.....	5
A. GROUND 1: Claims 1–5, 7–11, 13–16, 18 Are Obvious Over Henseler and Thyagarajan.....	5
1. Henseler (EX1004).....	5
2. Thyagarajan (EX1006).....	8
3. The Combination & Reasons to Combine	10
4. Reasonable Expectation of Success	14
5. Claim Element Analysis.....	15
B. GROUND 2: Claims 1-18 Are Obvious Over Henseler, Thyagarajan, and Kautzleben	53
1. Kautzleben (EX1007).....	54

2.	The Combination & Reasons to Combine	55
3.	Reasonable Expectation of Success	62
4.	Claim Element Analysis.....	63
VI.	PAYMENT OF FEES—37 C.F.R. §42.103.....	69
VII.	MANDATORY NOTICES—37 C.F.R § 42.8(a)(1)	70
A.	Real Party-In-Interest—37 C.F.R. § 42.8(b)(1).....	70
B.	Related Matters—37 C.F.R. § 42.8(b)(2)	70
C.	Lead And Back-Up Counsel—37 C.F.R. § 42.8(b)(3).....	71
D.	Service Information	71

LIST OF EXHIBITS

EX1001	U.S. Patent No. 7,640,332 to Sanjay Dahiya (“the ’332 patent”)
EX1002	The Prosecution History of the ’332 patent (“the Prosecution History”)
EX1003	Declaration and Curriculum Vitae of Dr. Bajaj
EX1004	US 2007/0168919 (Henseler)
EX1005	RESERVED
EX1006	US 7,721,277 (Thyagarajan)
EX1007	US 7,493,624 (Kautzleben)
EX1008	US Appl. Ser. No. 11/046,152
EX1009	BEAWebLogic Server, Developing Applications with WebLogic Server (Version 9.0) (October 6, 2006)
EX1010	US 7,051,316
EX1011	WO 2006/089411
EX1012	The JBoss Extensible Server, Proceedings International Middleware Conference, Fleury et al. (2003)
EX1013	George Thomas, “Introduction to the Internet Protocol,” <i>Auomation.com</i> (February 7, 2003)
EX1014	“TCP/IP Overview,” <i>Cisco</i> (August 10, 2005)
EX1015	U.S. Patent Publication No. 2003/0204645
EX1016	U.S. Patent No. 7,966,412

- EX1017 U.S. Patent No. 3,918,030
- EX1018 R. Fielding, *et al.*, “HyperText Transfer Protocol – HTTP/1.1,”
World Wide Web Consortium Request for Comments 2616
(June 1999)
- EX1019 Sun Developer Network (SDN) Site, JMX Adoption,
<https://web.archive.org/web/20060619170121/http://java.sun.com/products/JavaManagement/jmxadoption.html> (archived Jun. 19, 2006)
- EX1020 Sun Developer Network (SDN) Site, Java Management
Extensions (JMX) Overview,
<https://web.archive.org/web/20040404111240/http://java.sun.com/products/jmx/overview.html> (archived Apr. 4, 2004)
- EX1021 Sun Developer Network (SDN) Site, Java Management
Extensions (JMX) Technology Overview,
<https://web.archive.org/web/20060904234759/http://java.sun.com/j2se/1.5.0/docs/guide/jmx/overview/intro.html#wp997785>
(archived Sep. 4, 2006)

LISTING OF CLAIMS

Claim 1	
[1.pre]	A method for hot deployment and/or redeployment in grid computing environment, wherein the grid computing environment includes one or more grid nodes, comprising:
[1.a]	adding a new version of an application release bundle in a repository server;
[1.b]	determining by a discovery services module which of the one or more grid nodes are running an application associated with the added new version of the application release bundle upon adding the new version of the application release bundle in the repository server;
[1.c]	notifying a client application manager associated with one or more of the determined grid nodes about adding the new version of the application release bundle along with a type of data transfer protocol to use; and
[1.d]	hot deploying/redeploying the new version of the application release bundle on running one or more application servers in an associated grid node using an appropriate hot deployment plug-in based on the data transfer protocol by a respective one of the client application managers.
Claim 2	
[2.pre]	The method of claim 1, wherein hot deploying/redeploying the new version of the application release bundle on the running one or more application servers in the associated grid using the appropriate hot deployment plug-in based on the data transfer protocol by the respective client application manager, comprises:
[2.a]	selecting and invoking an appropriate hot deployment plug-in by a respective one of the client application managers based on the data transfer protocol to use for hot deploying/redeploying the new version of the application release bundle in an associated grid node; and

[2.b]	hot deploying/redeploying the new version of the application release bundle on running one or more application servers using the appropriate hot deployment plug-in in the associated grid node by the respective client application manager.
Claim 3	
[3.pre]	The method of claim 1, wherein hot deploying/redeploying the new version of the application release bundle on running one or more application servers using the appropriate hot deployment plug-in by the respective client application manager comprises:
[3.a]	hot deploying/redeploying the new version of the application release bundle on running one or more application servers using the appropriate hot deployment plug-in by a data transfer client associated with a respective one of the client application managers.
Claim 4	
[4]	The method of claim 1, wherein the application comprises a set of components that are hosted by one or more application servers in the one or more grid nodes.
Claim 5	
[5]	The method of claim 1, wherein the repository server stores one or more new versions of the application release bundle associated with the application.
Claim 6	
[6]	The method of claim 1, further comprising: registering the client application manager associated with one or more of the determined grid nodes with the repository server for accepting notification of receipt of the new version of the application release bundle.
Claim 7	
[7]	The method of claim 1, wherein the data transfer plug-in includes protocols selected from the group consisting of ftp, http, gridftp, tcp, and udp.

Claim 8	
[8]	The method of claim 1, wherein each grid node comprises one or more application servers that include components selected from the group consisting of Web containers, Web service containers, and EJB containers.
Claim 9	
[9.pre]	A computer-readable medium with program code embodied therein for causing when executed, a computer system, to perform a method for hot deployment/redeployment in a grid computing environment comprising:
[9.a]	adding a new version of an application release bundle in a repository server;
[9.b]	determining by a discovery services module which of the one or more grid nodes are running an application associated with the added new version of the application release bundle upon adding the new version of the application release bundle in the repository server;
[9.c]	notifying a client application manager associated with one or more of the determined grid nodes about adding the new version of the application release bundle along with a type of data transfer protocol to use; and
[9.d]	hot deploying/redeploying the new version of the application release bundle on running one or more application servers in an associated grid node using an appropriate hot deployment plug-in based on the data transfer protocol by a respective one of the client application managers.
Claim 10	
[10.pre]	The computer-readable medium of claim 9, wherein hot deploying/redeploying the new version of the application release bundle on the running one or more application servers in the associated grid using the appropriate hot deployment plug-in based on the data transfer protocol by the respective client application manager, comprises:

[10.a]	selecting and invoking an appropriate hot deployment plug-in by a respective one of the client application managers based on the data transfer protocol to use for hot deploying/redeploying the new version of the application release bundle in an associated grid node; and
[10.b]	hot deploying/redeploying the new version of the application release bundle on running one or more application servers using the appropriate hot deployment plug-in in the associated grid node by the respective client application manager.
Claim 11	
[11.pre]	The computer-readable medium of claim 9, wherein hot deploying/redeploying the new version of the application release bundle on running one or more application servers using the appropriate hot deployment plug-in by the respective client application manager comprises:
[11.a]	hot deploying/redeploying the new version of the application release bundle on running one or more application servers using the appropriate hot deployment plug-in by a data transfer client associated with a respective one of the client application managers.
Claim 12	
[12]	The computer-readable medium of claim 9, wherein the method further comprises: registering the client application manager associated with one or more of the determined grid nodes with the repository server for accepting notification of receipt of the new version of the application release bundle.
Claim 13	
[13]	The computer-readable medium of claim 9, wherein the data transfer plug-in includes protocols selected from the group consisting of ftp, http, gridftp, tcp, and udp.

Claim 14	
[14.pre]	A computer system comprising: a processing unit; and a memory coupled to the processing unit, the memory having stored therein code for causing, when executed, the processing unit to perform a method comprising:
[14.a]	adding a new version of an application release bundle in a repository server;
[14.b]	determining by a discovery services module which of the one or more grid nodes are running an application associated with the added new version of the application release bundle upon adding the new version of the application release bundle in the repository server;
[14.c]	notifying a client application manager associated with one or more of the determined grid nodes about adding the new version of the application release bundle along with a type of data transfer protocol to use; and
[14.d]	hot deploying/redeploying the new version of the application release bundle on running one or more application servers in an associated grid node using an appropriate hot deployment plug-in based on the data transfer protocol by a respective one of the client application managers.
Claim 15	
[15.pre]	The system of claim 14, wherein hot deploying/redeploying the new version of the application release bundle on the running one or more application servers in the associated grid using the appropriate hot deployment plug-in based on the data transfer protocol by the respective client application manager, comprises:
[15.a]	selecting and invoking an appropriate hot deployment plug-in by a respective one of the client application managers based on the data transfer protocol to use for hot deploying/redeploying the new version of the application release bundle in an associated grid node; and

[15.b]	hot deploying/redeploying the new version of the application release bundle on running one or more application servers using the appropriate hot deployment plug-in in the associated grid node by the respective client application manager.
Claim 16	
[16.pre]	The system of claim 14, wherein hot deploying/redeploying the new version of the application release bundle on running one or more application servers using the appropriate hot deployment plug-in by the respective client application manager comprises:
[16.a]	hot deploying/redeploying the new version of the application release bundle on running one or more application servers using the appropriate hot deployment plug-in by a data transfer client associated with a respective one of the client application managers.
Claim 17	
[17]	The system of claim 14, wherein the method further comprises: registering the client application manager associated with one or more of the determined grid nodes with the repository server for accepting notification of receipt of the new version of the application release bundle.
Claim 18	
[18]	The system of claim 14, wherein the application comprises a set of components that are hosted by one or more application servers in the one or more grid nodes.

CONVENTIONS OF THE PETITION

- All emphases are added unless noted otherwise;
- Bold-italic emphases correlate to claim language;
- Quotations are generally from exhibits, not claim language; and
- The phrase “as discussed” and equivalent phrases incorporate fully the analysis of the cross-cited portion of the Petition.

FedEx Corporation and FedEx Office and Print Services, Inc. petition for IPR of claims 1–18 (“Challenged Claims”) of U.S. Patent No. 7,640,332.

I. REQUIREMENTS FOR IPR—37 C.F.R. §42.104

A. Grounds for Standing—37 C.F.R. §42.104(a)

Petitioners certify that the ’332 patent is available for IPR, and that Petitioners are not barred or estopped from requesting IPR.

B. Challenge and Relief Requested—37 C.F.R. §42.104(b)

Petitioners request IPR and cancellation of the ’332 patent’s Challenged Claims based on the grounds below. Petitioners submit that the cited prior art references are themselves sufficient to establish obviousness pursuant to the analysis in Section V. The Section V analysis further cites to the Declaration of Dr. Bajaj for complementary corroboration and explanation. *See* EX1003.

Asserted Grounds		
ID	Claims	Prior Art Basis
1	1–5, 7–11, 13–16, 18	§ 103—Henseler (EX1004), Thyagarajan (EX1006)
2	1–18	§ 103—Henseler, Thyagarajan, Kautzleben (EX1007)

As shown below, each of the above identified references pre-dates the ’332 patent’s earliest claimed priority date of December 27, 2006. However, Petitioners do not concede that the ’332 patent is entitled to the claimed priority.

Reference	Prior Art Date	Status
Henseler (EX1004)	Filed Dec. 1, 2006	§102(e)
Thyagarajan (EX1006)	Filed Jun. 8, 2004	§102(e)
Kautzleben (EX1007)	Filed Dec. 30, 2003	§102(e)

II. THE '332 PATENT

A. Pertinent Disclosure

The '332 patent is directed to deploying new versions of applications within a grid computing environment in such a way that (1) keeps track of which nodes are running which applications, (2) considers data transfer protocols in deploying the applications, and (3) deploys the updated applications without shutting down or restarting an application container for the application. EX1001, 1:46-61, Claim 1, Figure 2; EX1003, ¶¶25–26. In particular, a new version of an application is “hot deployed,” which the '332 patent explains “refers to a process of deploying/redeploying an application without having to shutdown/restart an application container.” EX1001, 1:39–41.

An embodiment of the '332 patent is illustrated in Figure 2 (below). In this embodiment, a new version of an application release bundle is added to repository server 220, and upon adding the new version, discovery services module 280 determines which of the one or more grid nodes 235 are running the application.

EX1001, 3:61–4:7. The client application manager 230 associated with grid node 235 is notified regarding the new version, as well as a data transfer protocol to use. *Id.*, 4:8–36. Then, the client application manager hot deploys the new version of the application release bundle to the application servers 260 that are running the application. *Id.*, 4:37–5:19. The client application manager performs this hot deployment by using an appropriate hot deployment plug-in based on the data transfer protocol being used. *Id.*, 4:37–5:19.

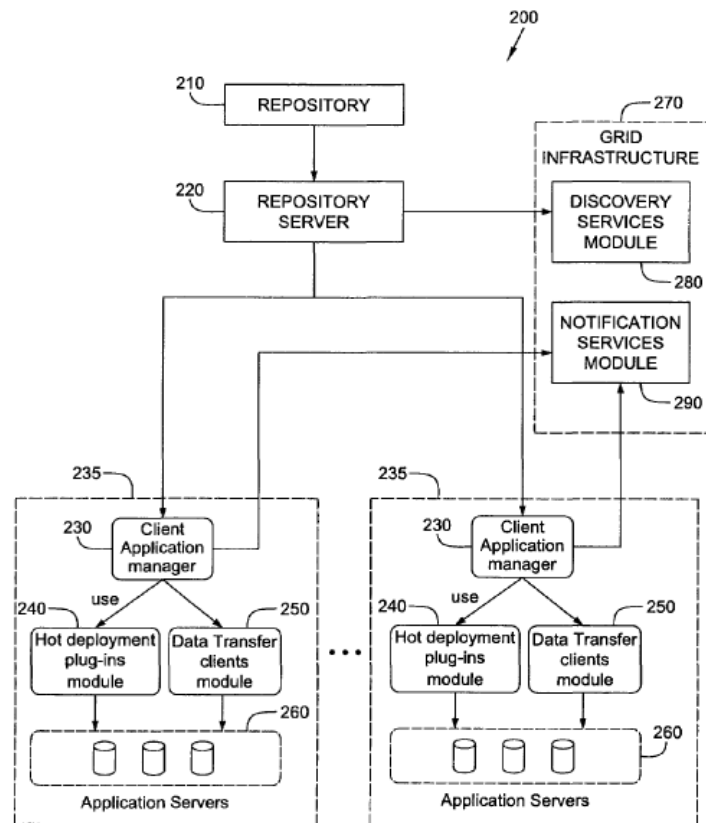


FIG. 2

EX1001, Figure 2.

B. Prosecution History

Hewlett-Packard Company filed the '332 patent on December 20, 2007, and it issued just two years later on December 29, 2009, pursuant to a first-action notice of allowance containing an examiner's amendment converting certain dependent claims into independent form. EX1002, 258–266. The notice of allowance does not provide any reason or rationale explaining how or why the claims purportedly distinguish over the nine prior art references cited by the examiner. *Id.*

III. LEVEL OF ORDINARY SKILL

A person of ordinary skill in the art (“POSITA”) relating to the subject matter of the '332 patent would have had (1) at least a master's degree in computer science, computer engineering, or a related field, and (2) at least two years of professional experience involving the study and/or development of distributed, clustered, and/or grid computing systems. EX1003, ¶¶28–29. Additional education could substitute for experience and vice versa. *Id.*

IV. CLAIM CONSTRUCTION—37 C.F.R. §42.104(b)(3)

The analysis below in Section V proves the Challenged Claims obvious under any interpretation reasonably advanced according to the *Phillips* standard. Petitioners submit that the Board need not resolve any express constructions. *See Wellman, Inc. v. Eastman Chem. Co.*, 642 F.3d 1355, 1361 (Fed. Cir. 2011). Should any subsequent discussion of claim interpretation arise here or in the parallel district

court proceeding, Petitioners reserve the right to address and respond to any construction advanced by Patent Owner, the Board, and/or the District Court.

V. THE CHALLENGED CLAIMS ARE UNPATENTABLE

A. GROUND 1: Claims 1–5, 7–11, 13–16, 18 Are Obvious Over Henseler and Thyagarajan

1. Henseler (EX1004)

Like the '332 patent, Henseler teaches a system for deploying and managing applications across a number of nodes in a grid computing environment. As illustrated in Figure 1, Henseler teaches a distributed computing system 10 that includes a number of application nodes, as well as a control node that manages the application nodes. EX1004, [0035]–[0050], [0199]–[0206], [0232]–[0238], Figures 1, 21, 24; EX1003, ¶¶31–33. The control node “manage[s] all aspects of the” grid, including deploying and “updating” applications. EX1004, [0182]; *see also id.*, [0142]. Henseler’s control node maintains a separate image repository, which stores instances of “application images” before they are deployed to application nodes. *Id.*, [0048]–[0049], [0109], [0142], [0236], [0238], Figure 1. Henseler describes deploying in a J2EE environment with JMX functionality. *Id.*, [0152], [0160], [0234], [0236]–[0237].

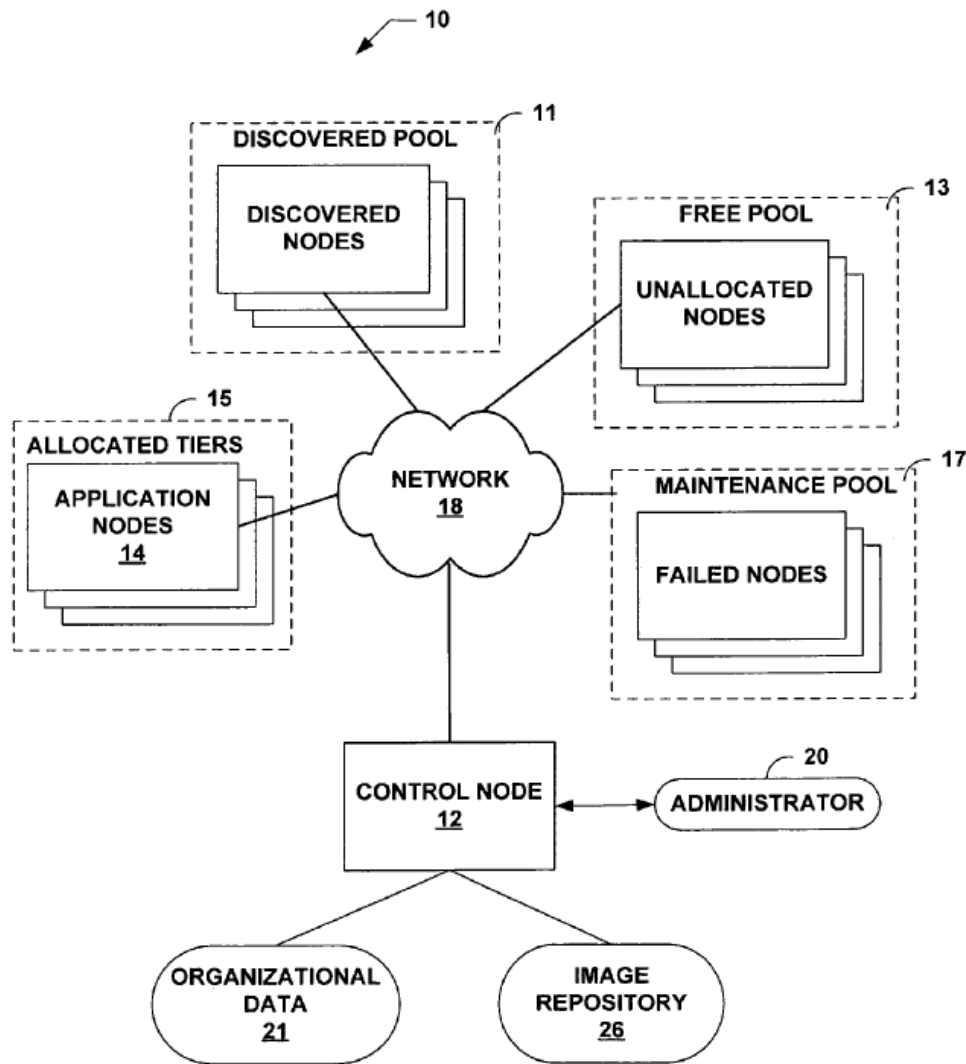


FIG. 1

EX1004, Figure 1.

As shown in Figure 21, the control node manages applications within the grid using several components, including (1) a monitoring subsystem, which collects information about the grid and its running applications, (2) an application service level automation infrastructure (“application SLAI”), which comprises an

application rules engine, (3) an application matrix, which includes configuration parameters for an application, and (4) an application governor, which receives deployment instructions from the application rules engine, and deploys applications according to configuration parameters included in the application matrix. EX1004, [0199]–[0238], Figures 1, 21, 22, 24.

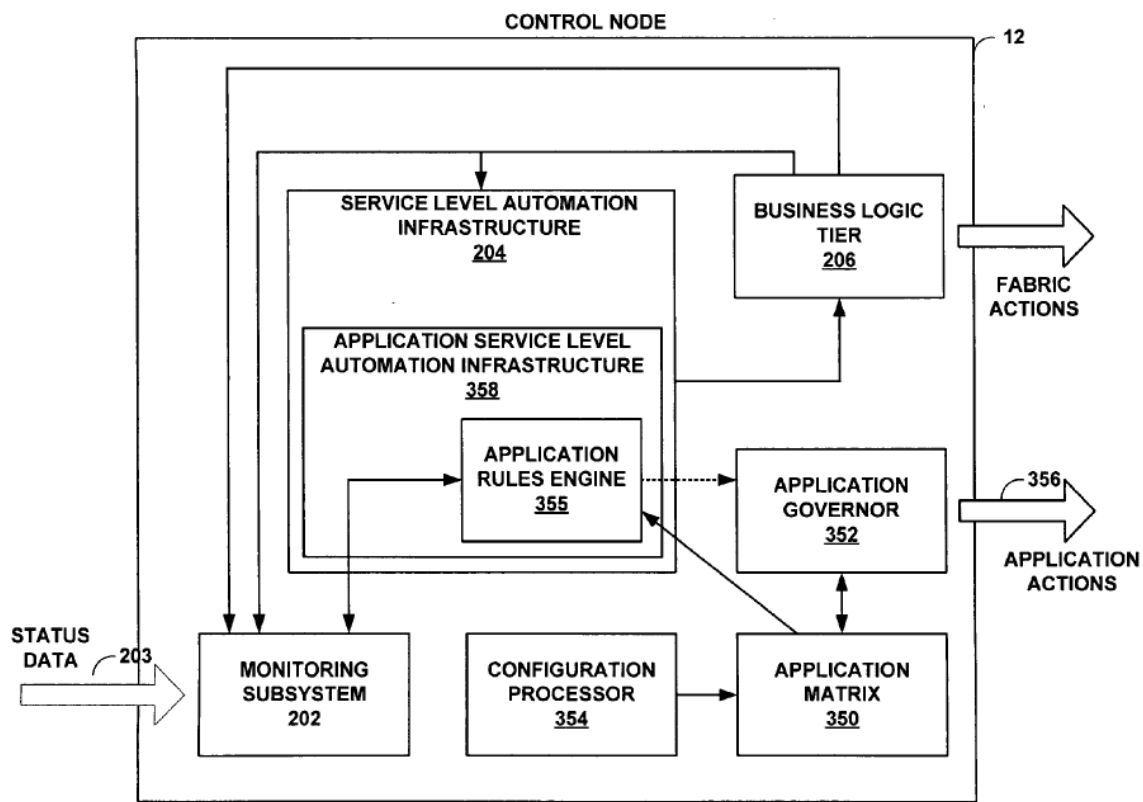
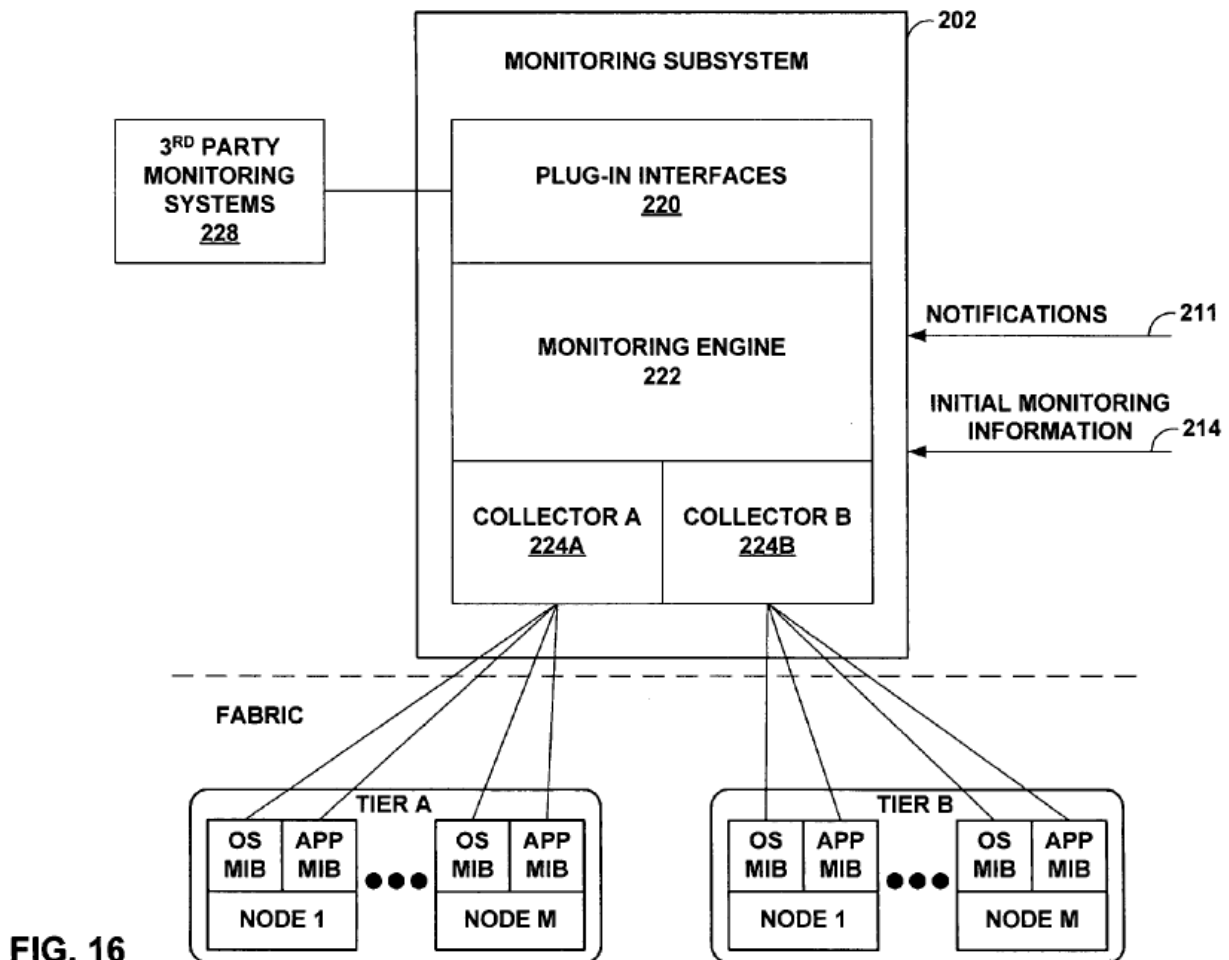


FIG. 21

EX1004, Figure 21.

As shown in Figure 16, a control node's monitoring subsystem includes a monitoring engine which receives collection requests, "and invokes the appropriate one of collectors 224 based on the protocol specified in the collection requests."

EX1004, [0158]. The monitoring subsystem “provides a plug-in architecture,” where “collectors 224 are responsible for protocol-specific collection of monitoring information.” *Id.*, [0157].



EX1004, Figure 16.

2. Thyagarajan (EX1006)

Thyagarajan is entitled “Hot Deployment of Shared Modules in an Application Server.” EX1006, Cover. Thyagarajan teaches hot deploying software

modules on J2EE application servers in order to decrease the system downtime and quicken development cycles in large enterprise deployments, like the one in Henseler. *Id.*, 1:5–26, 3:12–4:23; EX1003, ¶¶34–35. Notably, Thyagarajan emphasizes that “[i]n J2EE, for example, an EAR (J2EE enterprise archive) or WAR (J2EE web archive) are candidates for hot deployment in application servers as these applications are loaded by class loaders that are leaves in the application server loader class hierarchy.” *Id.*, 3:22–28. According to Thyagarajan, EAR and WAR files are particularly amenable to hot deployment, because “[t]hese applications are either self-contained or use classes that may be loaded by parent class loaders. There are no applications/components, dependent on these applications deployed in the application server instance.” *Id.*, 3:29–32. Thyagarajan explains that EAR and WAR files are hot deployed by “re-creating the class loader instance that represented the corresponding EAR or WAR and the new definitions of the classes in the EAR/WAR may be lazily loaded when they are first referred to.” *Id.*, 3:32–36.

Extending beyond EAR and WAR files, Thyagarajan also teaches how to hot deploy shared standalone resource adapter modules (RARs), which requires addressing more complex dependencies. *Id.*, 3:45–4:23. For example, Thyagarajan teaches “hot-deploy[ing] a different version of a module deployed on [an] application server” by undeploying an associated module class loader and redeploying to the class loader chain a different version of the module class loader

to load classes of the different module version. *Id.*, 12:61–13:1. By deploying a different version of a module in this manner, the “application server 300 does not need to be restarted for new versions of the classes to be loaded by the redeployed module class loader.” *Id.*, 13:1–9.

3. The Combination & Reasons to Combine

The Henseler-Thyagarajan Combination represents the predictable and expected result of applying in the context of Henseler’s technique for automated application deployment in a distributed computing system Thyagarajan’s teachings on hot deployment of applications and modules. EX1003, ¶36; *see also id.*, ¶¶37–48. From a POSITA’s perspective in 2006, integrating the supplemental teachings of Thyagarajan with Henseler’s disclosure would have involved nothing more than applying known solutions to known problems in the field of distributed computing systems to achieve predictable outcomes. *Id.* The fundamental aspects of Henseler’s disclosure would remain intact, bolstered by the well-known and beneficial teachings of Thyagarajan to arrive at a Combination within the scope of the ’332 patent’s claimed invention with a reasonable expectation of success. EX1003, ¶36.

The following independent reasons, taken separately or collectively, support the Henseler-Thyagarajan Combination. EX1003, ¶¶37–48.

First, a POSITA would have been prompted to pursue a combination with Thyagarajan based on Henseler’s suggestions to do so. EX1003, ¶¶38–39. For

example, Henseler teaches that a control node is responsible for managing the software images running on application server nodes within the distributed computing system, that these software images include “application files,” and that the control node can later “upgrad[e]” the applications. EX1004, [0045]–[0046], [0182]. Next, Henseler explains how the control node has the ability “to deploy new instances of a software application during execution of the software application,” which is consistent with a POSITA’s understanding of hot deployment. EX1004, [0243]; *see also id.*, [0237] (discussing application deployment “while distributed computing system 10 is operating”).

Interpreting Henseler’s teaching to deploy new software application instances “during execution” as a suggestion of hot deployment, a POSITA would have sought out and applied confirmatory teachings on this subject from references in the field of distributed computing, like Thyagarajan. EX1003, ¶39. Indeed, it was generally known in the art that hot deployment was beneficial because it facilitates application deployment on a large scale with reduced (or minimal) human intervention, which further promotes Henseler’s desire for a control node that provides “autonomic control” over the applications deployed throughout a large, distributed enterprise network. EX1004, Abstract, [0007], [0045], [0180]–[0182], [0199]–[0200], [0231], [0243]–[0244] (“obviate the need for manual configuration”), EX1003, ¶39.

Second, and as an additional suggestion of hot deployment that would have prompted the combination with Thyagarajan, Henseler discloses that the applications being deployed include “Java 2 Enterprise Edition (J2EE) application[s].” EX1004, [0234]. J2EE applications were common and gaining popularity by the time of the ’332 patent in 2006–07, and hot deployment was one of their well-known and desirable attributes. EX1003, ¶40. The ’332 patent expressly concedes this point:

In recent years, grid computing with web services for components oriented systems, such as J2EE based applications is gaining popularity. The J2EE based applications are built as a set of components. Generally, these components are hosted by a container or a server, which provides an execution environment to the components. In such a system, multiple applications may be hosted in a single container. In such a scenario, upgrading an application or hot redeployment of an application in a container does not require restarting the container as J2EE based container support hot deployment mechanisms. This is a very desirable feature as it would not impact running of other applications residing in the system.

EX1001, 1:28–39.

Third, Thyagarajan teaches that the “the ability to dynamically load”—or “hot deploy”—application components is a “require[ment]” for enterprise developers for the following reasons:

Lower system downtime: Large enterprise deployments of business components need to stay “alive” and reduce downtime to provide better service to users of the system and still work with modified implementations of business components.

Faster development cycles: Developers would like to see changes to a component without restarting the application server. Hot deployment of self-contained components in a system such as a J2EE application server may help to accelerate development cycles.

EX1006, 1:6–26. In short, Thyagarajan teaches that hot deployment was a known solution to known problems that provides predictable and beneficial results, including reduced downtime and accelerated application development. *Id.*; EX1003, ¶41. Given the similarities between Thyagarajan and Henseler—e.g., a shared focus on distributed computing environments and corresponding Java technology, such as J2EE applications, and EAR (J2EE enterprise archive) files—a POSITA would have reasonably expected the benefits espoused in Thyagarajan to apply with similar or equal effect in the context of Henseler. *Compare* EX1004, [0035]–[0042], Figures 1–2, *with* EX1006, 1:10–15, 12:39–13:9, Figure 7; *also compare* EX1004, [0152], [0159], [0219], [0234], [0236], *with* EX1006, 1:10–5:10, 7:49–55, 12:39–13:8; EX1003, ¶42.

Fourth, and relatedly, a POSITA would have pursued the Henseler-Thyagarajan combination based on the unique advantage Thyagarajan’s disclosure provides by extending the desirable hot deployment functionality beyond independent “EAR (J2EE enterprise archive) or WAR (J2EE web archive)” application components, as was already known, and to further include “RARs (Resource adapter modules),” which tended to “present problems” when these modules were “shared by multiple applications deployed in the application server instance.” EX1006, 3:13–4:23; *see also id.*, 5:14–6:20 (“Embodiments of a system and method for the hot deployment of shared modules,” “such as standalone resource adapter modules (RARs)[.]”), 12:39–14:2; EX1003, ¶¶43–44. In effect, Thyagarajan describes an approach where a shared RAR application component can be upgraded from an “old version” to a “new version” via a hot deployment. EX1006, 3:13–4:23, 5:14–6:20, 12:39–14:2; EX1003, ¶45. As discussed, given the commonality between Henseler and Thyagarajan regarding the use of Java technology (J2EE) in a distributed computing environment, a POSITA would have expected Thyagarajan’s beneficial technique to similarly apply in Henseler. *Id.*, ¶45.

4. Reasonable Expectation of Success

A POSITA would have reasonably expected success in the above-discussed Henseler-Thyagarajan Combination. EX1003, ¶¶46–48. As discussed at Sections V.A.1–3, hot deploying applications in the context of distributed computing

environments was well-known by the time of the '332 patent in 2006–07. *Id.* These technologies had been thoroughly described in the technical and patent literature, and even successfully demonstrated in the real-world by then. *Id.* Thus, the result of the Combination would have been predictable to a POSITA, and the POSITA would have expected it to work. *Id.*

5. Claim Element Analysis

(a) Claim 1

Element [1.pre]: A method for hot deployment and/or redeployment in grid computing environment, wherein the grid computing environment includes one or more grid nodes, comprising:

The Henseler-Thyagarajan Combination renders obvious Element [1.pre]. EX1003, ¶¶49–56. For example, Henseler teaches a “distributed computing system” comprising one or more control nodes, each managing a pool of application nodes that can be “shared between tiers and domains...to dynamically address high-processing demands, failures and other events.” EX1004, [0035]–[0042], [0064]–[0065] (“computing as a service”); *see also id.*, [0046], [0054] (“web server, application server”), [0058], [0108]–[0109] (discussing application nodes and node slots “belonging to [a] web server tier” and “provid[ing] web server functions”), Figure 1. Matching a POSITA’s understanding of a ***grid computing environment***, the nodes of Henseler’s distributed computing system “operat[e] in cooperation with each other to provide distributed processing.” EX1004, [0035]; EX1003, ¶50.

Moreover, Henseler specifically references well-known elements of a *grid computing environment*, such as “Java 2 Enterprise Edition (J2EE) application[s]” and the “web service definition language (WSDL).” EX1004, [0234], [0178]; EX1003, ¶51. Thus, a POSITA would have appreciated that Henseler’s teachings regarding the distributed computing system render obvious the claimed *grid computing environment*. EX1003, ¶51; *see also* EX1001, 1:28–37 (the ’332 patent conceding that, “[i]n recent years, grid computing with web services for components oriented systems, such as J2EE based applications is gaining popularity”).

Henseler teaches that the “distributed computing system 10 includes at least one control node 12, but may utilize additional control nodes to assist with the management functions. Other control nodes 12 (not shown in FIG. 1) are optional and may be associated with a different subset of the computing nodes within distributed computing system 10.” EX1004, [0041]–[0042]; EX1003, ¶52. Each control node “provides the system support functions for managing distributed computing system 10,” including “manag[ing] the roles of each computing node within distributed computing system 10 and the execution of software applications within the distributed computing system.” EX1004, [0041].

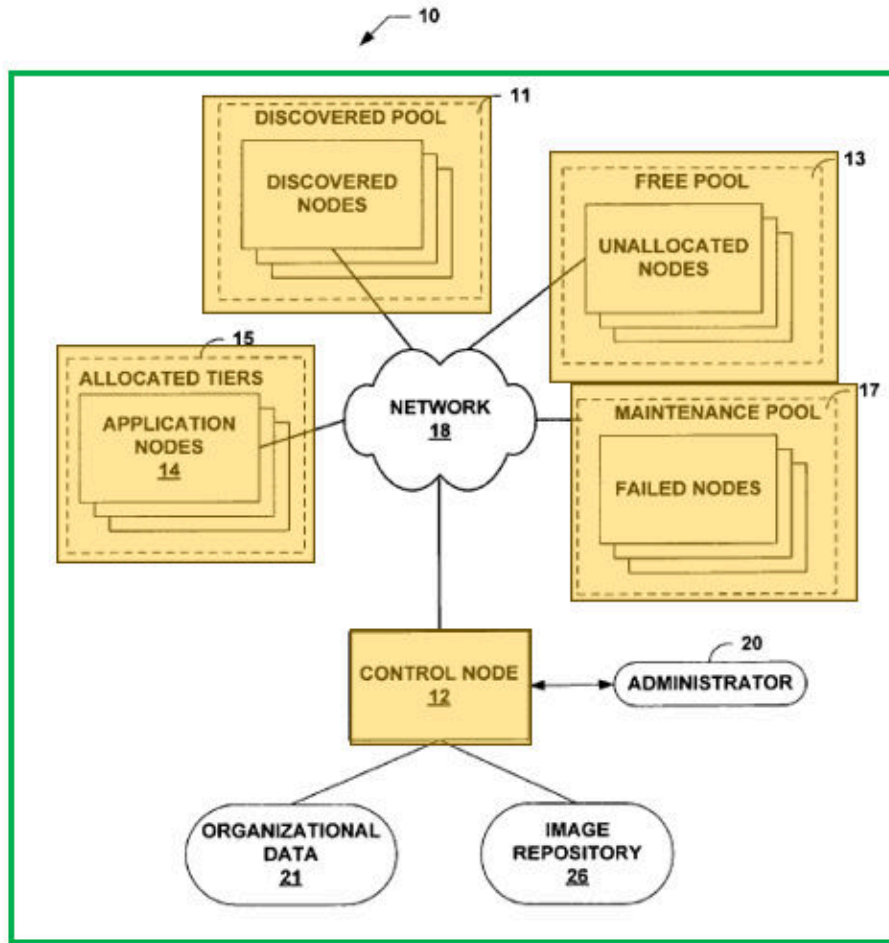


FIG. 1

EX1004, Figure 1 (annotating a grid node in orange, and outlining the grid computing environment in green). As annotated above, a control node and the set of application nodes managed by the control node, collectively, form a functional unit (*grid node*) within the distributed computing system (*grid computing environment*). EX1003, ¶53. Thus, Henseler's distributed computing system includes one or more *grid nodes*, the number depending on how many control nodes

and corresponding sets of managed application nodes are used in a particular implementation. *Id.*, ¶53.

The Henseler-Thyagarajan Combination further renders obvious a *method for hot deployment and/or redeployment* in the distributed computing system (*grid computing environment*). EX1003, ¶54. For one, Henseler teaches deploying “Java 2 Enterprise Edition (J2EE) application[s],” which were known to be hot-deployable. EX1004, [0234]; *see also id.*, [0045]–[0049], Figure 1; *supra* §§V.A.1, V.A.3; EX1003, ¶54. Henseler also teaches that a control node has the ability “to deploy new instances of a software application during execution of the software application,” which is consistent with a POSITA’s understanding of *hot deployment*. EX1004, [0243]; *see also id.*, [0237] (discussing application deployment “while distributed computing system 10 is operating”); EX1003, ¶54.

Building on Henseler’s teachings, Thyagarajan explicitly discloses that “[h]ot deployment of modules is typically supported for modules that are loaded by class loaders that are ‘leaves’ in an application server’s class loader hierarchy.” EX1006, 3:22–24. Thyagarajan gives examples of these “leaves” that would be typically amenable to hot deployment. “In J2EE, for example, an EAR (J2EE enterprise archive) or WAR (J2EE web archive) are candidates for hot deployment in application servers as these applications are loaded by class loaders that are leaves in the application server class hierarchy...Hot deployment of such components may

be achieved by re-creating the class loader instance that represented the corresponding EAR or WAR and the new definitions of the classes in the EAR/WAR may be lazily loaded¹ when they are first referred to.” *Id.*, 3:24–36. Thus, Thyagarajan teaches that J2EE applications are hot-deployable, and a POSITA would have understood the same. EX1003, ¶55.

To summarize, Henseler teaches the notion of deploying “Java 2 Enterprise Edition (J2EE) applications” during execution, and Thyagarajan confirms that J2EE applications are hot-deployable. EX1003, ¶56. Accordingly, the Henseler-Thyagarajan Combination implements a *method for hot deployment* in the context of Henseler’s distributed computing system (*grid computing environment*). EX1003, ¶56; *see also supra* §V.A.3–4.

Element [1.a]: adding a new version of an application release bundle in a repository server;

The Henseler-Thyagarajan Combination renders obvious Element [1.a]. EX1003, ¶¶57–63. For example, Henseler teaches that a control node’s configuration manager (part of its application governor) invokes a method to “create an instance of [an] application image,” and then “store[s] the instance of the

¹ In Thyagarajan, “lazy loading” refers to classes being “loaded on demand.” EX1006, 1:64-67.

application image in software image repository 26” (*repository server*). EX1004, [0238]. In this context, a POSITA would understand Henseler to teach that the configuration manager sends an instruction to the image repository to store the instance of the application image, and the instance is added to the repository server upon executing the instruction (*adding a new version of an application release bundle in a repository server*). EX1003, ¶57.

The application image “constitutes software code needed to execute the application,” and an instance of the image “is a separate copy the application image.” EX1004, [0236], [0238]; *see also id.*, [0049] (explaining that “[e]ach image instance represents a collection of bits that may be deployed on an application node”), [0219], [0233]–[0234] (explaining that “an ‘application image’ is a bitwise image of the software application at a time when the software application is ready to be executed”), Figure 22. As would have been understood and/or obvious to a POSITA, Henseler’s application image instance is an executable collection or *bundle* of bits that includes the components of the application and is used for the purpose of deploying or *releasing* an *application—an application release bundle*. EX1003, ¶¶58–59. Additionally, a POSITA would have been motivated to further package (*bundle*) the application image instance as an archive file to facilitate storage and hot deployment, as was conventional and described by Thyagarajan. *Id.*, ¶60.

The POSITA would also have appreciated that Henseler's software image *repository* provides and/or renders obvious the claimed *repository server*. EX1003, ¶61. As shown and described throughout Henseler, the software image repository is a distinct computing component of the distributed computer system for storing software and application images and image instances. EX1004, [0048]–[0049], [0142], [0236], [0238], Figure 1. It is likewise clear from Henseler that the software image repository performs the service of receiving, storing, and *serving* software and application images and image instances for the control node[s]. *Id.*

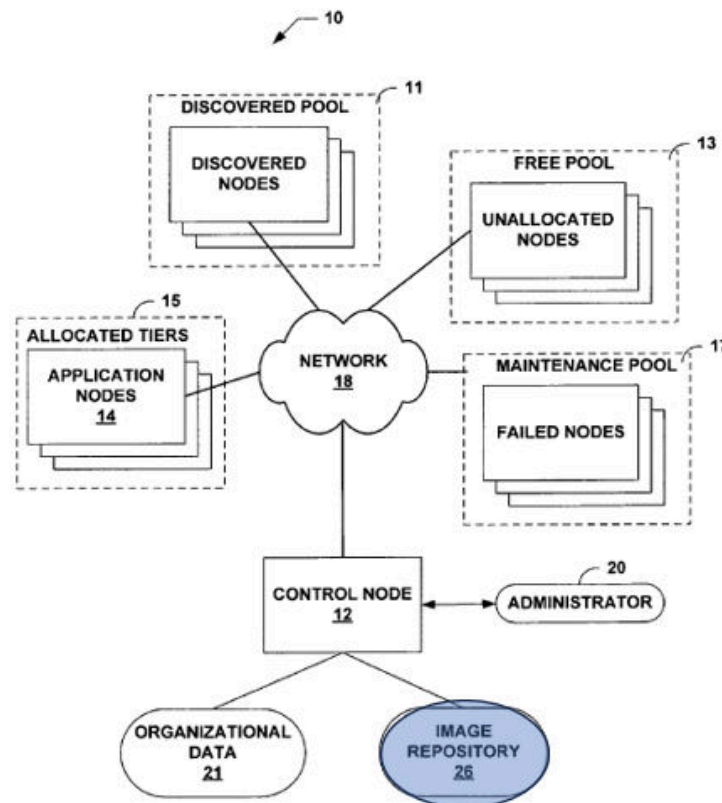


FIG. 1

EX1004, Figure 1 (annotating the repository server in blue).

Henseler further teaches that images within the software image repository can have different *versions*. *E.g.*, EX1004, [0142] (“the *version* of the image”). A POSITA would have appreciated that this rudimentary teaching applies to both software images/instances and application images/instances (*an application release bundle*). EX1003, ¶62. To wit, Henseler starts by explaining that a software image—which “refers to a complete set of software loaded on an individual computing node to provide an execution environment for one or more applications”—“may include application files.” EX1004, [0045]. And Henseler later discloses the notion of “updating...applications,” which the POSITA would have understood to mean deploying a *new version* of one or more application files from the initially loaded software image. *Id.*, [0182]; EX1003, ¶62.

Moreover, the Henseler-Thyagarajan Combination renders obvious *adding a new version of an application release bundle* in view of Thyagarajan’s teachings about hot deploying “a different *version* of a module deployed on [an] application server.” EX1006, 12:61–13:9; *see also id.*, 5:14–38 (discussing “hot deployment of shared modules in systems including application servers” and how “[w]hen a different *version* of the module is redeployed, a new class loader for the modules is added”). In Thyagarajan, consistent with a POSITA’s knowledge and understanding, the concept of module hot deployment/redeployment broadly applies to *applications* deployed via “an EAR (J2EE enterprise archive) or WAR (J2EE web

archive),” as well as “standalone RARs (Resource adapter modules).” *Id.*, 3:12–36; EX1003, ¶63; *see also* EX1004, [0219] (identifying “.ear” files in the context of services launched with deployed J2EE applications), [0222], Figure 22; EX1001, 1:28–45 (discussing the well-known fact that “J2EE based applications” “support hot deployment”), 2:53–55 (explaining that the “application release bundle can include application archive code”).

Element [1.b]: determining by a discovery services module which of the one or more grid nodes are running an application associated with the added new version of the application release bundle upon adding the new version of the application release bundle in the repository server;

The Henseler-Thyagarajan Combination renders obvious Element [1.b]. EX1003, ¶¶64–70.

Henseler teaches that the control node 12 includes, among other things, an application governor 352, an application matrix 350, and an application rules engine 355. *See* EX1004, [0199]–[0206], Figure 21. The application governor 352 includes a configuration manager 404 that “creat[es] an instance of the application image using the ‘createImageInstance’ method” and stores the application image instance in the software image repository. *Id.*, [0238], Figure 24. Henseler goes on to explain that, “[a]fter configuration manager 404 creates [and stores] at least one instance of the application image, application rules engine 355 may determine that an application node in the tier associated with the application should execute the

instance of the application image.” *Id.*, [0239]. In view of these disclosures, and for at least the following four reasons, a POSITA would have understood Henseler to teach and/or render obvious ***determining by a discovery services module which of the one more grid nodes are running an application associated with the added new version of the application release bundle upon adding the new version of the application release bundle in the repository server.*** EX1003, ¶65.

First, as discussed above and at Element [1.a], the combined disclosures of Henseler and Thyagarajan with respect to storing a created application image instance in the repository server teaches and/or renders obvious ***adding the new version of the application release bundle in the repository server.*** EX1003, ¶66.

Second, Henseler teaches that “after” (*upon*) ***adding*** the application image instance (***new version of the application release bundle***) by the configuration manager, the application rules engine “***determine[s]***” ***which*** application node (part of the ***grid node***, per Element [1.pre]) should execute that instance. EX1004, [0239]. Henseler does not teach or suggest any intermediate operations between the ***adding*** step and the ***determining*** step. EX1003, ¶67. And, indeed, a POSITA would have appreciated that there is no reason for a temporal or causal delay between them, particularly in view of Henseler’s emphasis on the importance of “efficient and automated” operations. EX1004, [0005]; *see also id.*, [0047]; EX1003, ¶67. Accordingly, it would have been apparent and/or obvious to the POSITA that

Henseler’s *determination* step occurs *upon adding the new version of the application release bundle*. *Id.*, ¶67.

Third, as would have been understood and/or obvious to a POSITA, Henseler’s application rules engine is a *modular* element within the control node that ascertains (*discovers*) the prescribed applications and *services* of the application nodes via the application matrix and its working memory—*discovery services module*. EX1004, [0168], [0172], [0176], [0177], [0193], [0195]–[0198], [0200], [0203], [0204], [0221]–[0222], [0227], [0230]–[0231], [0238]–[0239], Figures 20, 21, 22; EX1003, ¶68.

Fourth, Henseler teaches that the application *node determined* by the application rules engine (*discovery services module*) to execute the application image instance is part of a “tier *associated* with the application.” EX1004, [0239]. In the context of “updating...applications,” a POSITA would have understood this disclosure to mean that the determined/targeted node is *running* an older version of *the application*. *Id.*, [0182]; *see also id.*, [0045] (software images include “application files”); EX1003, ¶69; *supra* Element [1.a] (explaining that Henseler and Thyagarajan contemplate updating an existing application with a new version via hot deployment/redeployment).

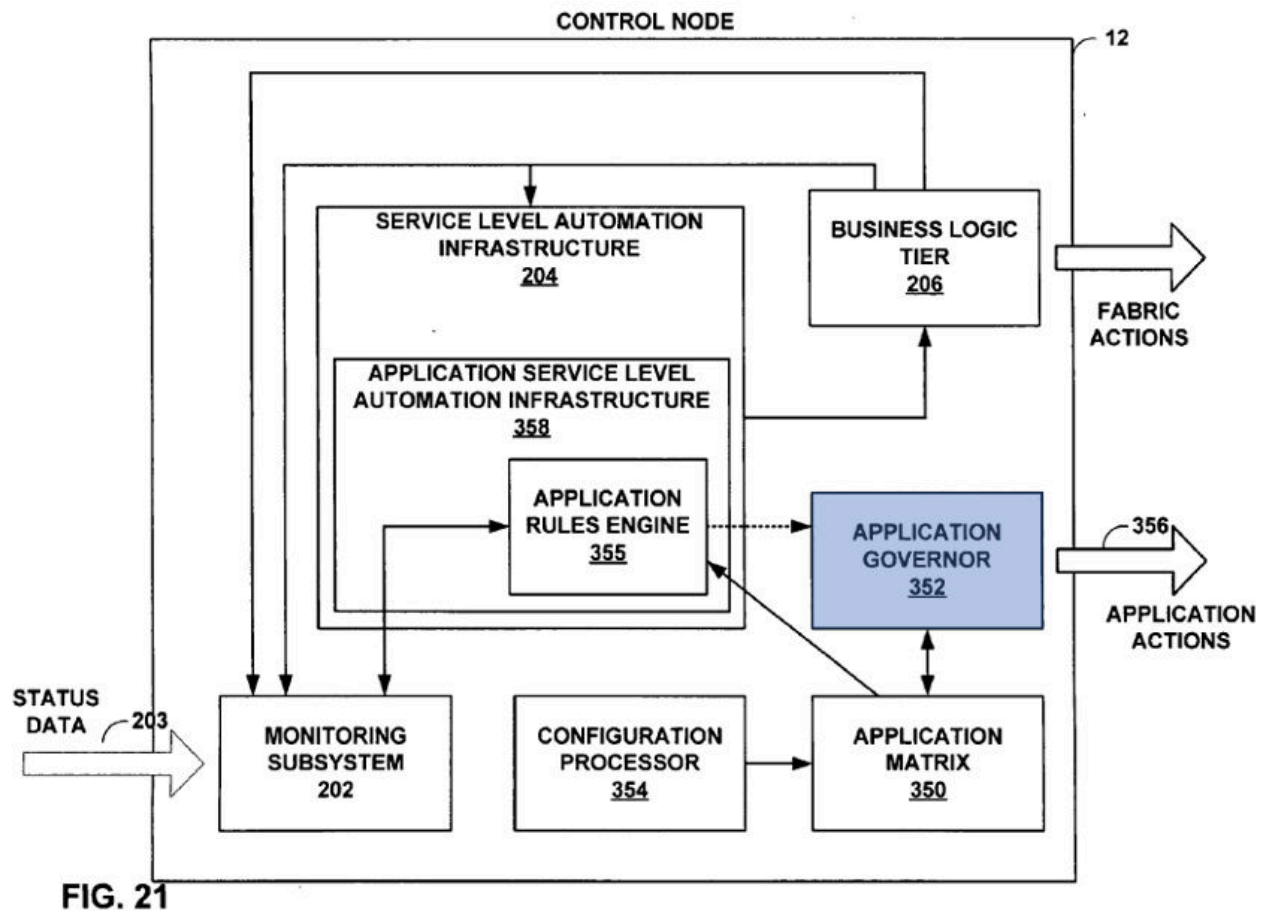
To the extent Patent Owner contests this interpretation of Henseler, it would have been exceedingly obvious to implement Henseler’s teaching to

“updat[e]...applications” (EX1004, [0182]) using the approach of deploying a new version to the *nodes* currently *running* an older version of a particular *application*. EX1003, ¶70. For one, a POSITA would have been motivated to use this approach because it merely follows the teachings and suggestions within Henseler. *Id.*, ¶70 (citing EX1008). Moreover, a POSITA would have understood that hot deploying a new version of an application image instance on nodes already running the application would improve efficiency. *Id.*, ¶70 (explaining how the application rules engine would “easily determine” the application nodes based on information captured in the application matrix). Additionally, by hot deploying on nodes already running the application, the application rules engine’s task of determining where to deploy new versions of the application is made simpler. *Id.*, ¶70 (explaining the conservation of “time and resources” achieved by leveraging application nodes that have already been deemed suitable for running the application, leaving other nodes unburdened and available for “other functions or applications”).

Element [1.c]: notifying a client application manager associated with one or more of the determined grid nodes about adding the new version of the application release bundle along with a type of data transfer protocol to use; and

The Henseler-Thyagarajan Combination renders obvious Element [1.c]. EX1003, ¶¶71–79. To start, Henseler describes a “software engine” called an application governor 352 that “performs application-level actions based on requests

received from application rules engine 355” and, thus, *manages* the “deployment and monitoring of applications” on *client application* nodes. EX1004, [0203]; *see also id.*, [0199]–[0200], Figure 21; EX1003, ¶71.



EX1004, Figure 21 (annotating the client application manager in blue).

Next, it would have been apparent and/or obvious to a POSITA that Henseler’s application governor 352 (*client application manager*), which is part of the control node 12 (*associated with one or more of the determined grid nodes*, per Elements [1.pre]/[1.b]), is *notified about adding* an application image instance (*new*

version of the application release bundle, per Element [1.a]) as part of the process flow when the application rules engine 355 issues deployment instructions. EX1003, ¶72. For example, Henseler teaches that, when application rules engine 355 “detect[s] that a node...is ready,” it “directs application governor 352 to handle the details of deploying the application to the node. In turn, application governor 352 accesses application matrix 350 to retrieve application-specific parameters necessary to deploy the application.” EX1004, [0204]–[0205] (“Application governor 352 then uses the discovered configuration parameters to communicate with the applications.”); *see also id.*, [0238]–[0239]. In more detail, Henseler explains how application governor 352 includes a configuration manager 404 that receives instructions from application rules engine 355 to deploy an application image instance, and then invokes a “configureContainerResource” method,² which “appl[ies] a set of configuration parameters defined in application matrix 350...to configure a container resource on the application node to host the instance of the application image.” EX1004, [0239].

² This method is separate and distinct from the “createImageInstance” method, which is invoked by the configuration manager 404 to create instances of an application image. EX1004, [0238]; EX1003, ¶72, n.3.

From a POSITA's perspective, this aspect of Henseler's disclosure teaches and/or renders obvious that application governor 352 (*client application manager*) is *notified about adding* an application image instance (*new version of the application release bundle*). EX1003, ¶73. For one, by receiving the deployment instruction from the rules engine, the application governor is *notified* that the system is ready to facilitate deployment—confirming that the application image instance has been successfully *added* to the image repository, an application node has been determined for running the image instance, and both the image instance and node are ready for deployment. EX1003, ¶73. Additionally, the application governor is further *notified about the added* application image instance when the governor accesses the application matrix to discover the necessary information to carry out deployment. EX1003, ¶73.

In addition, a POSITA would have recognized Henseler's further teachings that the information in the application matrix discovered by the application governor 352 in response to a deployment instruction by the application rules engine 355 (*notifying*) includes *a type of data transfer protocol to use* for deployment. EX1003, ¶74. For example, the application matrix entry for a given application contains information in an “application nodes column 362,” such as an IP address for each of the “tier node slots to which the application may be assigned.” EX1004, [0211]; *see also id.*, [0218] (“a list of all of the available tier-node slots to which

control node 12 may deploy application 374”), [0223], Figure 22. The matrix entry also includes “an application column 360 that includes elements generally related to the deployment of the application,” such as the IP address and port number for “the server and port [that] hosts the administrative part of the logically defined application.” EX1004, [0210], [0217], [0223], Figure 22. As a POSITA would have known, an IP address is an identifier that uniquely identifies a device on a network employing the Internet Protocol, a longstanding and widely used network-level *protocol* for *transferring data* packets. EX1003, ¶74; *see also* EX1004, [0043] (explaining that control node 12 communicates with other nodes over a communications network 18), Figure 1.

Additionally (and alternatively), a POSITA would have appreciated Henseler’s disclosure that a “user, such as a system administrator, specifies the protocol or *protocols to be used* for monitoring a software image when the image is created.” EX1004, [0160]–[0161] (identifying “[e]xample protocols,” such as “Simple Network Management Protocol (SNMP), “Hypertext Transfer Protocol (HTTP),” and “Secure Shell (SSH)” protocol); *see also id.*, [0157]–[0158] (“existing and new protocols”). Moreover, the POSITA would have been motivated to apply this protocol specification teaching in the context of Henseler’s related disclosures on application deployment for at least the following reasons (taken individually or collectively). EX1003, ¶75; *see also Bos. Sci. Scimed, Inc. v. Cordis Corp.*, 554

F.3d 982, 991 (Fed. Cir. 2009) (“Combining two embodiments disclosed adjacent to each other in a prior art patent does not require a leap of inventiveness.”).

First, a POSITA would have been prompted to combine Henseler’s above-discussed teachings based on a suggestion to do so provided by the fact that an application matrix entry specifies various IP (Internet Protocol) addresses. EX1003, ¶76; EX1004, [0210]–[0211], [0217]–[0218], [0223], Figure 22. A POSITA would have known that the Internet Protocol was a longstanding and widely used network-level protocol that controls how data packets are routed and delivered across the Internet. EX1003, ¶76. The POSITA would have likewise understood that higher level protocols are typically built or stacked on top of the Internet Protocol, including application-layer protocols like those Henseler describes with respect to software image monitoring (e.g., HTTP, SNMP, SSH). *Id.*, ¶76. Thus, Henseler’s disclosure of the network-level protocol information in the application matrix, would have suggested and motivated a POSITA to pursue incorporation of Henseler’s additional disclosure to further specify the applicable application-layer protocol(s). *Id.*, ¶76.

Second, Henseler also suggests and motivates this combination of teachings by explaining in the “monitoring” context that “collecting information from operating systems and applications executing on nodes” is performed by “protocol-specific” plug-ins. EX1004, [0156]–[0162]. Given that collecting information from applications and deploying applications both require communications across a

network between the control node and the application node[s], a POSITA would have recognized that Henseler's teachings to specify a suitable application-level protocol and use "protocol-specific" plug-ins would obviously apply in both contexts. EX1003, ¶77. Indeed, it is true in both the monitoring/collecting context and the deployment context that different types of applications and nodes in different domains and tiers of a distributed computing environment can be designed to communicate according to different protocols. *Id.*; *see also* EX1004, [0051]–[0058] ("web server functions, business application functions and database functions"), [0106]–[0109], [0243] ("[T]he embodiment may provide a common, pluggable framework for supporting diverse software applications."), Figure 2; EX1003, ¶78. When, as here, "a technique has been used to improve one [aspect of a] device, and a person of ordinary skill in the art would recognize that it would improve similar [aspects] in the same way, using the technique is obvious unless its actual application is beyond his or her skill." *KSR Int'l Co. v. Teleflex Inc.*, 550 U.S. 398, 417 (2007).

Further still, in the context of Henseler's teachings regarding the autonomic control of applications, the motivated POSITA would have naturally incorporated the specified application-level protocol information in the application matrix, such that it would be discovered by the application governor (***notifying a client application manager***) during deployment and monitoring, just like all of the other deployment and configuration details. EX1003, ¶79. Analogous to specifying

configuration information for monitoring software images, Henseler similarly teaches that such information is likewise specified for monitoring and deploying applications and held in “an application matrix entry” of the application matrix 350. See EX1004, [0207]–[0223] (discussing the configuration and deployment information of the application matrix), [0224]–[0229] (describing a process for creating an application matrix entry), [0232]–[0237] (explaining how configuration information gleaned when creating application images and associated objects is added to the application matrix), Figures 21–24.

Element [1.d]: hot deploying/redeploying the new version of the application release bundle on running one or more application servers in an associated grid node using an appropriate hot deployment plug-in based on the data transfer protocol by a respective one of the client application managers.

The Henseler-Thyagarajan Combination renders obvious Element [1.d]. EX1003, ¶¶80–85.

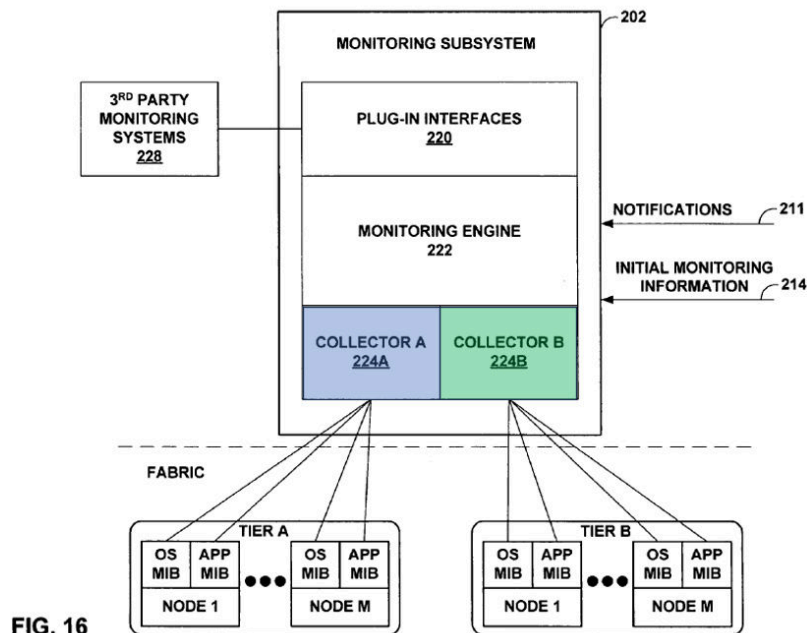
As discussed throughout Elements [1.pre]–[1.c], the combined teachings of Henseler and Thyagarajan render obvious the concept of using a control node to ***hot deploy a new version*** of an application image instance (***application release bundle***) on one or more ***application server*** nodes within an ***associated grid node*** that are already ***running*** an older version of the same application. Moreover, and as a POSITA would have known, the teachings of Henseler and Thyagarajan make clear that hot deployment occurs dynamically, *e.g.*, while the application server nodes

remain up and **running**. See EX1004, [0243] (explaining that new application image instances are deployed “during execution of the software application”); EX1006, 1:10–26 (discussing, in the context of hot deployment, “the ability to dynamically load new implementations” of application components, meaning changes can be made “without restarting the application server”), 7:32–40 (explaining how application modules “may be dynamically redeployed...without affecting other modules or requiring the application server to be restarted”); EX1003, ¶81.

As discussed at Element [1.c], Henseler’s teachings render obvious the notion of **deploying** application image instances (**application release bundle**) by a **respective** application governor (**client application manager**) within the control node that uses a **data transfer protocol** (e.g., a network-level protocol, transport-level protocol, and/or an application-level protocol) discovered via an entry of the application matrix. E.g., EX1004, [0157]–[0158], [0160]–[0161], [0207]–[0229], [0232]–[0237], Figure 22. Moreover, for the following reasons, it would have been obvious in view of Henseler’s teachings for the application governor to use an **appropriate**, protocol-specific **plug-in** to facilitate deployment. EX1003, ¶82.

To start, as discussed at Element [1.c], Henseler expressly discloses in the “monitoring” context that “collecting information from operating systems and applications executing on nodes” is performed by “protocol-specific” **plug-ins** called “collectors.” EX1004, [0156]–[0162]. Different collectors can easily be

“*plugged-in*” for existing and new protocols and for existing and new hardware and software.” *Id.*, [0057], [0159] (explaining that the collectors are simply “Java code...loaded with a class loader” and can be “dynamically configured”). As shown in Henseler’s Figure 16 (below), the collectors 224 are part of a monitoring subsystem 202 within the control node 12. *Id.*, [0157]. The subsystem 202 further includes a monitoring engine 222 that receives collection requests “and invokes the appropriate one of collectors 224 based on the protocol specified in the collection requests.” *Id.*, [0158]. This modular, “*plug-in* architecture allows for new protocols to be added by simply adhering to a collector *plug-in* signature,” making the system extendable, adaptable, and flexible. *Id.*, [0157]; *see also id.*, [0055] (touting “a high degree of flexibility”); EX1003, ¶83.



EX1004, Figure 16 (annotating protocol-specific collectors in blue and green).

Again, as discussed at Element [1.c], given that collecting information from applications and deploying applications both require communications across a network between the control node and the application node[s], a POSITA would have recognized that Henseler's teachings to specify a suitable application-level protocol and use "protocol-specific" plug-ins would obviously apply in both contexts. EX1003, ¶84. Indeed, it is true in both the monitoring/collecting context and the deployment context that different types of applications and nodes in different domains and tiers of a distributed computing environment can be designed to communicate according to different protocols. *Id.*; see EX1004, [0051]–[0058] ("web server functions, business application functions and database functions"), [0106]–[0109], [0243] (discussing support for "diverse software applications" in a distributed environment where different parties, such as vendors, consultants, and partners, may write new applications), Figure 2.

Henseler makes clear that protocol-specific plug-ins are a solution to this technical problem, and a POSITA would have understood that plug-ins were well known by the time of the '332 patent in 2006–07 and had been used for decades to extend the functionality of existing systems in a similar way. EX1003, ¶84. In addition, the "plug-in architecture" is a common theme that permeates Henseler and is explicitly applied to the application governor for controlling deployment. See EX1004, Abstract ("A pluggable, application-specific application governor is

selected...to control deployment, execution, and monitoring of the applications[.]”), [0167] (discussing “plug-in classes” in the context of a sensor subsystem), [0201]–[0202] (explaining that the “configuration processor” that “generates an application matrix entry” is “pluggable”), [0206] (“[A]pplication governor 352 is also ‘pluggable.’”), [0243] (emphasizing the “advantages” of “provid[ing] a common, pluggable framework for supporting diverse software applications”).

To summarize, because:

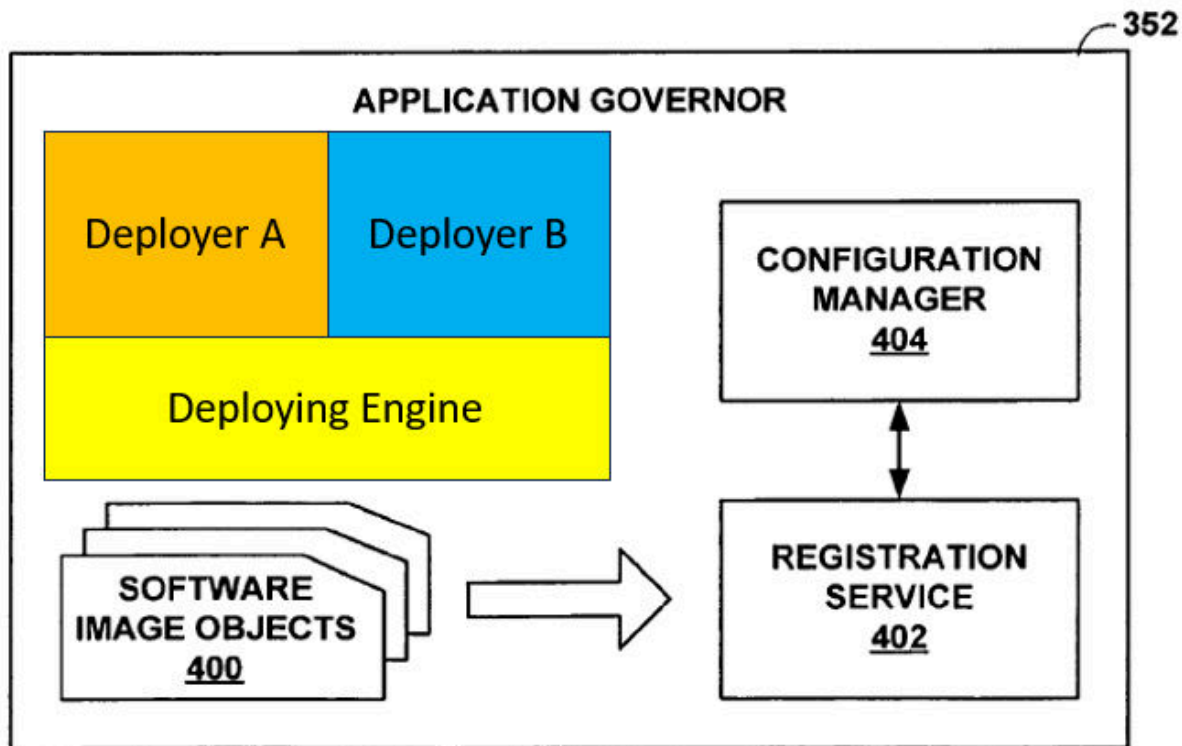
(i) Henseler expressly discloses protocol-specific plug-ins for monitoring and collecting information from diverse applications;

(ii) a POSITA would have appreciated that monitoring/collecting and deployment similarly involve communicating with applications and nodes via protocols; and

(iii) Henseler promotes a plug-in architecture generally and in the context of the application governor,

a POSITA would have been motivated to implement Henseler’s application governor with multiple protocol-specific ***deployment plug-ins***, where the appropriate plug-in is selected and invoked based on the protocol information in the application matrix that is associated with the application image being deployed. EX1003, ¶85. Just as Henseler’s monitoring subsystem 202 provides a plug-in architecture with a monitoring engine for selecting and invoking the appropriate

protocol-specific plug-in to facilitate monitoring communications with a given application server node, it would have been obvious to configure Henseler's application governor 352 to deploy applications in a substantially similar, protocol-specific way (*e.g.*, with a deploying engine for selecting and invoking protocol-specific deployers, as illustrated in modified Figure 24 below). EX1003, ¶85. This straightforward combination of Henseler's teachings would have predictably produced the very kind of beneficial result that Henseler envisioned—*i.e.*, a “pluggable framework for supporting diverse software applications.” EX1004, [0243]; EX1003, ¶85.



EX1004, Figure 24 (modified).

(b) Claim 2

Element [2.pre]: The method of claim 1, wherein hot deploying/redeploying the new version of the application release bundle on the running one or more application servers in the associated grid using the appropriate hot deployment plug-in based on the data transfer protocol by the respective client application manager, comprises:

Supra Element [1.d]. EX1003, ¶86.

Element [2.a]: selecting and invoking an appropriate hot deployment plug-in by a respective one of the client application managers based on the data transfer protocol to use for hot deploying/redeploying the new version of the application release bundle in an associated grid node; and

The Henseler-Thyagarajan Combination renders obvious Element [2.a]. EX1003, ¶87. As discussed at Element [1.d], a POSITA would have been motivated to implement Henseler's application governor (*client application manager*) with multiple protocol-specific *hot deployment* plug-ins and a deploying engine that *selects and invokes the appropriate plug-in based on* the protocol information in the application matrix (*data transfer protocol to use*) that is associated with the application image (*new version of the application release bundle*) being deployed. EX1003, ¶87. Henseler's monitoring subsystem is configured in this fashion for facilitating protocol-specific monitoring communications with a given application server node, and it would have been obvious to configure Henseler's application governor to conduct deployment communications in a substantially similar protocol-specific way. EX1003, ¶87; EX1004, [0152], [0156]–[0162], [0230], Figure 16.

Element [2.b]: hot deploying the new version of the application release bundle on running one or more application servers using the appropriate hot deployment plug-in in the associated grid node by the respective client application manager.

Supra Element [1.d]. EX1003, ¶88.

(c) **Claim 3**

Element [3.pre]: The method of claim 1, wherein hot deploying/redeploying the new version of the application release bundle on running one or more application servers using the appropriate hot deployment plug-in by the respective client application manager comprises:

Supra Element [1.d]. EX1003, ¶89.

Element [3.a]: hot deploying/redeploying the new version of the application release bundle on running one or more application servers using the appropriate hot deployment plug-in by a data transfer client associated with a respective one of the client application managers.

The Henseler-Thyagarajan Combination renders obvious Element [3.a]. EX1003, ¶¶90–92. As discussed at Element [1.d], a POSITA would have been motivated to implement Henseler’s application governor (***client application manager***) with multiple protocol-specific ***hot deployment*** plug-ins and a deploying engine that selects and invokes ***the appropriate plug-in*** based on the protocol information in the application matrix that is associated with the application image (***new version of the application release bundle***) being deployed. EX1003, ¶90; EX1004, Abstract, [0156]–[0162], [0167], [0201]–[0202], [0206], [0243], FIG. 24.

This deploying engine within the application governor renders obvious the claimed *data transfer client*. EX1003, ¶91. It facilitates communications (*data transfer*) with the application server nodes by selecting and invoking (*using*) the appropriate hot deployment plug-in; and it is a *client* in the sense that it performs this functionality based on protocol information served by the application matrix. EX1003, ¶91; EX1004, [0156]–[0162].

Various other aspects of Henseler’s application governor—such as the configuration manager 404 and application image object 400 (taken individually or collectively)—similarly render obvious the claimed *data transfer client* and, thus, provide additional/alternative bases for Element [3.a]. EX1003, ¶92; *see also* EX1004, [0007], [00203]–[0204], [0232], Figure 24. For example, Henseler explains how, “to deploy an instance of the application image to the application node,” the configuration manager 404 responds to instructions from the application rules engine 355 by retrieving from registration service 402 “a name of one of [the] software image objects 400 that is associated with the instance” and “us[ing] the name...to invoke a ‘configureContainerResource’ method...[and] a ‘startInterpretation’ method.” EX1004, [0239]; *see also id.*, [0232]–[0237]. The configureContainerResource method “configure[s] a container resource on the application node to host the instance of the application image” based on “a set of configuration parameters defined in the application matrix 350.” *Id.* The

startInterpretation method “instructs the container resource of the application node to start running the instance of the application image.” *Id.* In this way, as part of the modified version of Henseler discussed at Elements [1.c]–[1.d], the configuration manager and software image object produce deployment instructions (*data transfer*) and use the appropriate hot deployment plug-in to convey those instructions directed to a targeted application server node. EX1003, ¶92. The configuration manager and software image object are also *clients* in the sense that they retrieve and employ information from the registration service and the application matrix. EX1003, ¶92.

(d) **Claim 4**

4. The method of claim 1, wherein the application comprises a set of components that are hosted by one or more application servers in the one or more grid nodes.

The Henseler-Thyagarajan Combination renders obvious Claim 4. EX1003, ¶93. For example, in describing the “staging” process step of creating a new entry in the application matrix, Henseler teaches that a deployed *application* includes *a set of* “software *components*.” EX1004, [0224]–[0225]. Moreover, and as discussed at Elements [1.pre]–[1.d], Henseler makes clear that a deployed application is *hosted by one or more application servers in the one or more grid nodes*. EX1003, ¶93. For example, Henseler explains how, “to deploy an instance of the application image to the application node,” the configuration manager 404

within the application governor “invoke[s] a ‘configureContainerResource’ method...[and] a ‘startInterpretation’ method.” EX1004, [0239]; *see also id.*, [0232]–[0237]. The configureContainerResource method “configure[s] a container resource on the *application* node [server] to host the instance of *the application* image” based on “a set of configuration parameters defined in the application matrix 350.” *Id.* The startInterpretation method “instructs the container resource of the application node to start running the instance of the application image.” *Id.*

(e) **Claim 5**

5. The method of claim 1, wherein the repository server stores one or more new versions of the application release bundle associated with the application.

Supra Element [1.a]. EX1003, ¶94.

(f) **Claim 7**

7. The method of claim 1, wherein the data transfer plug-in includes protocols selected from the group consisting of ftp, http, gridftp, tcp, and udp.

The Henseler-Thyagarajan Combination renders obvious Claim 7. EX1003, ¶95. As discussed at Element [1.d], a POSITA would have been motivated to implement Henseler’s application governor with multiple protocol-specific deployment (*data transfer*) *plug-ins* and a deploying engine that selects and invokes the appropriate plug-in based on the protocol information in the application matrix that is associated with the application image being deployed. EX1003, ¶95. As

discussed at Elements [1.c]–[1.d], Henseler’s application matrix identifies the network-level IP (Internet Protocol) *protocol* and it would have been obvious to further include the application-level *protocols* Henseler elsewhere discloses, such as “Hypertext Transfer Protocol (*HTTP*).” EX1004, [0160]–[0161], [0211], Figure 22. As a POSITA would have known at the time of the ’332 patent in the 2006–07 timeframe, the HTTP protocol was built almost exclusively on the *TCP/IP* protocol stack. EX1003, ¶95. Accordingly, Henseler’s disclosures of the application-level and network-level protocols HTTP and IP additionally renders obvious the associated transport-level protocol *TCP*. *Id.*, ¶95.

(g) **Claim 8**

8. The method of claim 1, wherein each grid node comprises one or more application servers that include components selected from the group consisting of Web containers, Web service containers, and EJB containers.

The Henseler-Thyagarajan Combination renders obvious Claim 8. EX1003, ¶¶96–97. As discussed at Elements [1.pre]–[1.d], Henseler makes clear that applications are deployed on the *application servers* of a *grid node*. EX1003, ¶96. For example, Henseler explains how, “to deploy an instance of the application image to the application node,” the configuration manager 404 within the application governor “invoke[s] a ‘configureContainerResource’ method...[and] a ‘startInterpretation’ method.” EX1004, [0239]; *see also id.*, [0233] (explaining how

“software applications execute within ‘container resources,’” a term that “refers to an execution environment in which an application may execute”), [0235]. The configureContainerResource method “configure[s] a *container* resource on the *application* node [server] to host the instance of *the application* image” based on “a set of configuration parameters defined in the application matrix 350.” *Id.* The startInterpretation method “instructs the container resource of the application node to start running the instance of the application image.” *Id.* Given that Henseler discloses deploying applications on application nodes that “provide *web* server functions” and deploying applications that include “*EJBs*” (Enterprise Java Beans), it would have been apparent and/or obvious to a POSITA that the *containers* hosting those applications include *web containers* and *EJB containers*. EX1004, [0046], [0233], Figure 22; *see also id.*, [0054], [0058], [0068] (“application containers”), [0075] (“web application servers”), [0108]–[0109], [0217]–[0223], [0234] (discussing “Java 2 Enterprise Edition” applications), [0236] (discussing “Java Management Extension Management Bean (JMX MBean)”); EX1003, ¶¶96–97.

(h) **Claim 9**

Element [9.pre]: A computer-readable medium with program code embodied therein for causing when executed, a computer system, to perform a method for hot deployment/redeployment in a grid computing environment comprising:

The Henseler-Thyagarajan Combination renders obvious Element [9.pre]. EX1003, ¶¶98–99. For example, Henseler teaches each node in its distributed computing system “includes one or more programmable processors for executing software instructions stored on one or more computer-readable media.” EX1004, [0037]. Further, Henseler’s invention “is directed to a ***computer-readable medium*** comprising instructions. The instructions cause the processor to receive status data for a distributed computing system having a plurality of application nodes interconnected via a communications network. The status data represents an actual state of the application nodes. ... The instructions further cause the processor to process the status data with rules in a set of rule engines to determine operations for reducing any difference between an expected state and the actual state of the distributed computing system. The instructions cause the processor to maintain a set of application image objects associated with images of software applications. ... In addition, the instructions cause the processor to direct, in accordance with the determined operations, an application governor to deploy an image of a software application to one of the application nodes by invoking one or more of the methods

of the application image objects associated with the image of the software application.” *Id.*, [0009]; *see also id.*, Claim 20. Thus, in view of the instructions described above (e.g., for processing status data, conforming the actual state to the expected state for the distributed computing system, maintaining application image objects, and invoking those application image objects to deploy application images), Henseler teaches or renders obvious ***computer-readable medium with program code embodied therein for causing when executed, a computer system, to perform a method for deployment in a grid computing environment.*** EX1003, ¶98.

Moreover, as explained for Element [1.pre], the Henseler-Thyagarajan Combination renders obvious the same program code causing when executed, a computer system to perform a method for ***hot deployment*** in a grid computing environment. EX1003, ¶99.

Element [9.a]: adding a new version of an application release bundle in a repository server;

Supra Element [1.a]. EX1003, ¶100.

Element [9.b]: determining by a discovery services module which of the one or more grid nodes are running an application associated with the added new version of the application release bundle upon adding the new version of the application release bundle in the repository server;

Supra Element [1.b]. EX1003, ¶101.

Element [9.c]: notifying a client application manager associated with one or more of the determined grid nodes about adding the new version of the application release bundle along with a type of data transfer protocol to use; and

Supra Element [1.c]. EX1003, ¶102.

Element [9.d]: hot deploying/redeploying the new version of the application release bundle on running one or more application servers in an associated grid node using an appropriate hot deployment plug-in based on the data transfer protocol by a respective one of the client application managers.

Supra Element [1.d]. EX1003, ¶103.

(i) **Claim 10**

Element [10.pre]: The computer-readable medium of claim 9, wherein hot deploying/redeploying the new version of the application release bundle on the running one or more application servers in the associated grid using the appropriate hot deployment plug-in based on the data transfer protocol by the respective client application manager, comprises:

Supra Element [1.d]. EX1003, ¶104.

Element [10.a]: selecting and invoking an appropriate hot deployment plug-in by a respective one of the client application managers based on the data transfer protocol to use for hot deploying/redpeploying the new version of the application release bundle in an associated grid node; and

Supra Element [2.a]. EX1003, ¶105.

Element [10.b]: hot deploying/redeploying the new version of the application release bundle on running one or more application servers using the appropriate hot deployment plug-in in the associated grid node by the respective client application manager.

Supra Element [1.d]. EX1003, ¶106.

(j) **Claim 11**

Element [11.pre]: The computer-readable medium of claim 9, wherein hot deploying/redeploying the new version of the application release bundle on running one or more application servers using the appropriate hot deployment plug-in by the respective client application manager comprises:

Supra Element [1.d]. EX1003, ¶107.

Element [11.a]: hot deploying/redeploying the new version of the application release bundle on running one or more application servers using the appropriate hot deployment plug-in by a data transfer client associated with a respective one of the client application managers.

Supra Element [3.a]. EX1003, ¶108.

(k) **Claim 13**

13. The computer-readable medium of claim 9, wherein the data transfer plug-in includes protocols selected from the group consisting of ftp, http, gridftp, tcp, and udp.

Supra Claim 7. EX1003, ¶109.

(l) **Claim 14**

Element [14.pre]: A computer system comprising: a processing unit; and a memory coupled to the processing unit, the memory having stored therein code for causing, when executed, the processing unit to perform a method comprising:

The Henseler-Thyagarajan Combination renders obvious Element [14.pre]. EX1003, ¶110. For example, Henseler teaches each node in its distributed computing system “includes one or more programmable *processors* for executing software instructions stored on one or more computer-readable media.” EX1004,

[0037]. Further, Henseler's invention "is directed to a computer-readable medium comprising instructions. The instructions cause the processor to receive status data for a distributed computing system having a plurality of application nodes interconnected via a communications network. The status data represents an actual state of the application nodes. ... The instructions further cause the processor to process the status data with rules in a set of rule engines to determine operations for reducing any difference between an expected state and the actual state of the distributed computing system. The instructions cause the processor to maintain a set of application image objects associated with images of software applications. ... In addition, the instructions cause the processor to direct, in accordance with the determined operations, an application governor to deploy an image of a software application to one of the application nodes by invoking one or more of the methods of the application image objects associated with the image of the software application." *Id.*, [0009]; *see also id.*, Claim 20. Thus, in view of the instructions described above (*e.g.*, for processing status data, conforming the actual state to the expected state for the distributed computing system, maintaining application image objects, and invoking those application image objects to deploy application images), Henseler teaches or renders obvious ***a computer system comprising a processing unit, and a memory coupled to the processing unit, the memory having stored therein code for causing, when executed, the processing unit to perform a method.***

EX1003, ¶110. A POSITA would have understood Henseler to teach, or would have found it obvious, that the claimed code is *stored in memory* in Henseler's control node. EX1003, ¶110. Otherwise, the code would be unable to persist and be executed to cause the processing unit to perform the operations that Henseler teaches. *Id.*, ¶110.

Element [14.a]: adding a new version of an application release bundle in a repository server;

Supra Element [1.a]. EX1003, ¶111.

Element [14.b]: determining by a discovery services module which of the one or more grid nodes are running an application associated with the added new version of the application release bundle upon adding the new version of the application release bundle in the repository server;

Supra Element [1.b]. EX1003, ¶112.

Element [14.c]: notifying a client application manager associated with one or more of the determined grid nodes about adding the new version of the application release bundle along with a type of data transfer protocol to use; and

Supra Element [1.c]. EX1003, ¶113.

Element [14.d]: hot deploying/redeploying the new version of the application release bundle on running one or more application servers in an associated grid node using an appropriate hot deployment plug-in based on the data transfer protocol by a respective one of the client application managers.

Supra Element [1.d]. EX1003, ¶114.

(m) Claim 15

Element [15.pre]: The system of claim 14, wherein hot deploying/redeploying the new version of the application release bundle on the running one or more application servers in the associated grid using the appropriate hot deployment plug-in based on the data transfer protocol by the respective client application manager, comprises:

Supra Element [1.d]. EX1003, ¶115.

Element [15a]: selecting and invoking an appropriate hot deployment plug-in by a respective one of the client application managers based on the data transfer protocol to use for hot deploying/redeploying the new version of the application release bundle in an associated grid node; and

Supra Element [2.a]. EX1003, ¶116.

Element [15.b]: hot deploying/redeploying the new version of the application release bundle on running one or more application servers using the appropriate hot deployment plug-in in the associated grid node by the respective client application manager.

Supra Element [1.d]. EX1003, ¶117.

(n) Claim 16

Element [16.pre]: The system of claim 14, wherein hot deploying/redeploying the new version of the application release bundle on running one or more application servers using the appropriate hot deployment plug-in by the respective client application manager comprises:

Supra Element [1.d]. EX1003, ¶118.

Element [16.a]: hot deploying/redeploying the new version of the application release bundle on running one or more application servers using the appropriate hot deployment plug-in by a data transfer client associated with a respective one of the client application managers.

Supra Element [3.a]. EX1003, ¶119.

(o) **Claim 18**

18. The system of claim 14, wherein the application comprises a set of components that are hosted by one or more application servers in the one or more grid nodes.

Supra Claim 4. EX1003, ¶120.

B. GROUND 2: Claims 1-18 Are Obvious Over Henseler, Thyagarajan, and Kautzleben

Kautzleben supplements Henseler and Thyagarajan with respect to certain rudimentary features regarding communication and synchronization between distributed system components. Integration of Kautzleben's teachings does not disturb the aspects of Henseler and Thyagarajan mapped to Dependent Claims 2–5, 7–8, 10–11, 13, 15–16, and 18 in Ground 1. To avoid repetition, the analysis of these dependent claims is not repeated below because it is identical to Ground 1 and fully incorporated here in Ground 2. From Ground 1, only Independent Claims 1, 9, and 14 are addressed below. Dependent Claims 6, 12, and 17, which are not challenged in Ground 1, are also addressed below based on Kautzleben's complementary teachings. EX1003, ¶121.

1. Kautzleben (EX1007)

Kautzleben is entitled “Management Architecture and Method Employed Within a Clustered Node Configuration.” EX1007, Cover. Kautzleben teaches a “monitoring system which simplif[ies] the management of complex, multi-tiered networks such as those used in large enterprises,” like the one taught in Henseler. EX1007, 2:36–38; EX1003, ¶¶122–123. Kautzleben teaches using MBeans (Java objects representing JMX [Java Management Extensions] manageable resources) to implement a monitoring system for a cluster. EX1007, 2:36–50, 4:5–9. For example, Kautzleben teaches monitoring resources³ associated with a cluster using “resource MBeans” and mapping these MBeans to “monitor MBeans,” which receive updates regarding the monitored resources. *Id.*, 8:30–9:39. The resource MBean may notify a monitor MBean by “push[ing] the monitoring data to the monitor service using a specified event mechanism.” *Id.*, 9:40–45.

As shown below in Figure 12, after detecting a triggering event in a resource, a resource MBean 625 generates a notification to the notification service 1201, which “provides the notification to any objects that are registered with the notification service 1201 as listeners,” such as MBean 605. EX1007, 12:1–10.

³ “Virtually any type of system resource may be monitored,” according to Kautzleben’s teachings. EX1007, 6:56–65.

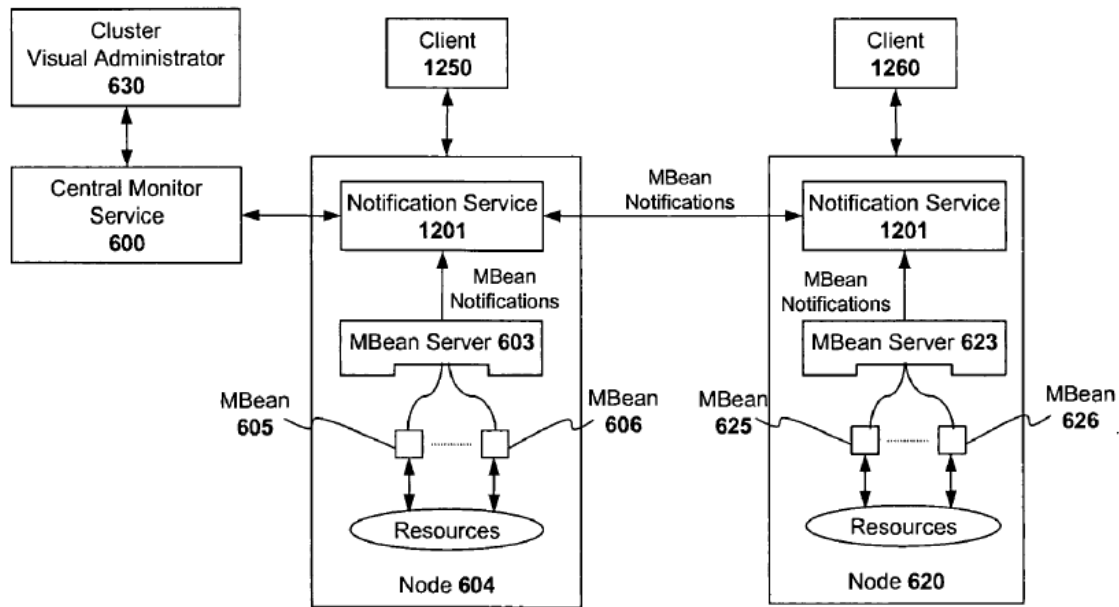


Fig. 12

EX1007, Figure 12.

2. The Combination & Reasons to Combine

The Henseler-Thyagarajan-Kautzleben Combination represents the predictable and expected result of applying in the context of Henseler's technique for automated application deployment in a distributed computing system, and Thyagarajan's technique for hot deployment of applications and modules, Kautzleben's teachings about monitoring resources and components in an environment using managed Java objects called "MBeans." EX1003, ¶124; *see also id.*, ¶¶125–138. From a POSITA's perspective in the 2006–07 timeframe of the '332 patent, integrating the supplemental teachings of Kautzleben with Henseler's and

Thyagarajan's disclosures would have involved nothing more than applying known solutions to known problems in the field of distributed computing systems to achieve predictable outcomes. *Id.*, ¶125.

Kautzleben describes a technique that employs well-established Java technologies—specifically, MBeans and notification services—to facilitate communication and synchronization between distributed components such as Henseler's application governor and image repository. EX1003, ¶125. Mechanisms like Kautzleben's were widely understood and routinely used in enterprise Java systems to support modular, event-driven coordination. *Id.* Given that Henseler's architecture already relies on similar technologies to manage distributed application deployment (as explained further below), incorporating Kautzleben's approach would have been a straightforward and technically sound extension. *Id.* In fact, a POSITA would have recognized this integration as a natural refinement of Henseler's teachings to address the interaction between the application governor and image repository. *Id.* The fundamental aspects of the Henseler-Thyagarajan Combination presented in Ground 1 would remain intact, bolstered by the well-known and beneficial teachings of Kautzleben to arrive at a Henseler-Thyagarajan-Kautzleben Combination embraced by the scope of the '332 patent's claimed invention with a reasonable expectation of success. *Id.*, ¶125.

The following independent reasons, taken separately or collectively, support the Henseler-Thyagarajan-Kautzleben Combination. EX1003, ¶¶126–138.

First, a POSITA would have been prompted to pursue a combination with Kautzleben’s teachings based on Henseler’s multiple suggestions to do so. EX1003, ¶¶127–132. For example, Henseler emphasizes that a control node fulfills its management role in the distributed computing system by collecting monitoring information and updating various components based on the “actual state” of constituent application nodes. EX1004, [0008]–[0009] (“actual state of the application nodes”), [0152]–[0177] (“actual state of the fabric defined for the organizational model implemented by [the] distributed computing system”), [0182] (“managing all aspects of the fabric,” including “adding, viewing, updating, and removing...notifications,...connectors, and monitors”). Notably, Henseler uses the same “Java Management Extensions” (JMX) technology for “collect[ing] and export[ing] real-time monitoring information” between the control node and application nodes that Kautzleben employs and seeks to further improve. EX1004, [0152], [0160]; EX1007, 3:49–4:30, 11:19–51, 14:25–28. Henseler also uses this shared “JMX MBean server” technology within the control node’s application governor 352 to implement a registration service 402 that allows a deployment configuration manager 404 to send a command that invokes the various deployment-

related “method[s]” of application image objects 400. EX1004, [0236]–[0241] (“registration service 402 may be a JMX MBean server”), Claim 9.

To summarize, Henseler employs the same kind of JMX MBean server technology as Kautzleben for enabling operative communications between the control node and various application nodes and between components within the control node’s application governor. EX1003, ¶128. Accordingly, when Henseler teaches that the application governor’s configuration manager 404 “store[s a created] instance of the application image in software image repository 26,” a POSITA would have been motivated to implement such an interaction utilizing the same kind of JMX MBean server technology that Kautzleben describes and that Henseler employs elsewhere for providing command instructions. EX1004, [0238]; EX1003, ¶¶128–129; *see also supra*, §V.A.5 (Element [1.a]).

Moreover, just as Henseler and Kautzleben also teach leveraging the JMX MBean server technology for “monitoring” nodes, the POSITA would have been motivated to employ the same technique for monitoring the image repository. *E.g.*, EX1004, [0152]; EX1007, 6:48–55. Indeed, because the application governor provides storage instructions to the image repository, it would have been apparent to the POSITA that the application governor is one of the “objects interested in receiving notifications” from the repository, *e.g.*, to confirm execution of those instructions and to otherwise remain synchronized. EX1007, 12:27–36; *see also id.*,

5:10–20 (discussing “synchronization” between server application instances), 14:49-51 (“provid[ing] messaging and synchronization services between the server nodes”); EX1003, ¶130. This approach is consistent with Henseler’s multiple other teachings about “notifications” passed between inter-dependent components of the control node and furthers Henseler’s desire for the control node to enable “autonomic control” over the applications deployed throughout a large, distributed enterprise network. *See* EX1004, [0155] (“BLT 206 outputs notifications 211 to SLAI 204 and monitoring subsystem 202”), [0165] (further discussing “notifications 211”), [0170] (“trigger listeners 247 [] receive sensor notifications”), [0171] (“rules contain notification calls”), [0173], [0196] (describing a “rule for outputting a notification”), [0200] (explaining how “application matrix 350 sends an alert whenever application matrix 350 changes”), [0229] (similar), [0231] (similar); *see also id.*, Abstract, [0007] (“The control node...provide[s] autonomic control of the application nodes[.]”), [0045], [0180]–[0182], [0199]–[0200], [0231], [0243]–[0244]; EX1003, ¶131.

Moreover, just as Kautzleben’s notification-based approach parallels several aspects of Henseler’s system, the concept of using notifications to coordinate interactions—such as those between Henseler’s application governor and the image repository—directly supports Henseler’s broader objective of enabling “autonomic control” over the deployment and management of applications across a large,

distributed enterprise network. EX1004, Abstract, [0007] (“The control node...provide[s] autonomic control of the application nodes[.]”), [0045], [0180]–[0182], [0199]–[0200], [0231], [0243]–[0244]; EX1003, ¶132.

Second, a POSITA would have recognized that combining Kautzleben with Henseler-Thyagarajan is merely a predictable combination of known software elements using the same underlying object-oriented programming language—Java—to solve a common technical problem regarding communication and synchronization between inter-dependent system components. EX1003, ¶¶133–135. In more detail, a POSITA reading Henseler’s teaching that the application governor’s configuration manager 404 “store[s a created] instance of the application image in software image repository 26” would have appreciated that the governor and repository require a means of communicating and remaining synchronized. EX1004, [0238]; EX1003, ¶133; *see also supra*, §V.A.5 (Element [1.a]). Seeking a solution to this straightforward technical problem that is compatible with Henseler’s Java-based architecture, the POSITA would have sought out and pursued the teachings of supplemental references like Kautzleben directed to communication and synchronization services based on the Java Management Extensions (JMX) server technology. EX1003, ¶133; EX1007, 3:49-4:30, 11:39-51, 14:20-33; *see also id.*, 6:56–65 (“Virtually any type of system resource may be monitored in this manner.”). Kautzleben’s JMX/MBean-based solution would have been particularly

desirable because its integration would merely require the addition of relatively simple MBeans and MBean servers, leaving the fundamentals of the Henseler-Thyagarajan Combination intact. EX1003, ¶133.

Third, a POSITA would have been motivated to pursue Kautzleben’s specific teachings to supplement the Henseler-Thyagarajan Combination because of the known benefits attributed to the JMX MBean server technology that Kautzleben employs. EX1003, ¶¶136–138. For example, it was known in the art that using an MBean server to facilitate communication and synchronization between components provided certain advantages over using direct inter-component communications—*e.g.*, remote management/monitoring capabilities, standardized access and interaction via the JMX MBean specification, increased modularity and flexibility. EX1003, ¶136. As discussed, Kautzleben teaches that its monitoring system “simplif[ies] the management of complex, multi-tiered networks such as those used in large enterprises.” EX1007, 2:36–38. Kautzleben’s monitoring system is primarily disclosed as improving the management of large enterprise networks in “a JMX implementation within a J2EE environment.” *See, e.g., id.*, 14:25–28. Similarly, Henseler and Thyagarajan are both concerned with managing large enterprise networks, in which applications are deployed in a J2EE environment, and where monitoring may be performed through JMX. EX1004, [0152], [0234]; EX1006, 1:10–20, 3:12–65, 12:39–45. Given the similarities between Henseler,

Thyagarajan, and Kautzleben—*e.g.*, a shared focus on distributed computing environments in large enterprise networks and corresponding Java technology, such as J2EE applications and monitoring through JMX—a POSITA would have expected the benefits associated with Kautzleben to apply with similar or equal effect in the context of the Henseler-Thyagarajan Combination. EX1003, ¶137.

3. Reasonable Expectation of Success

A POSITA would have reasonably expected success in the above-discussed Henseler-Thyagarajan-Kautzleben Combination. EX1003, ¶138. As discussed at Sections V.B.1–2, incorporating MBeans and MBean servers in J2EE environments, like the one taught in Henseler, was well-known by the time of the '332 patent in 2006–07. *Id.*, ¶138. Indeed, Henseler even teaches conforming the naming of its application image objects to a JMX MBean naming convention, and implementing the application governor's registration service as a JMX MBean server, from which a POSITA would understand that MBean functionality could easily be incorporated into Henseler's components. EX1004, [0236]–[0237]; EX1003, ¶138. These technologies had been thoroughly described in the technical and patent literature, and even successfully demonstrated in the real-world by then. *Id.*, ¶138. Thus, the result of the Combination would have been predictable to a POSITA, and the POSITA would have expected it to work. *Id.*, ¶138.

4. Claim Element Analysis

(a) Claim 1

Element [1.pre]: A method for hot deployment and/or redeployment in grid computing environment, wherein the grid computing environment includes one or more grid nodes, comprising:

Supra Ground 1, Element [1.pre]. EX1003, ¶139.

Element [1.a]: adding a new version of an application release bundle in a repository server;

Supra Ground 1, Element [1.a]. EX1003, ¶140.

Element [1.b]: determining by a discovery services module which of the one or more grid nodes are running an application associated with the added new version of the application release bundle upon adding the new version of the application release bundle in the repository server;

Supra Ground 1, Element [1.b]. EX1003, ¶141.

Element [1.c]: notifying a client application manager associated with one or more of the determined grid nodes about adding the new version of the application release bundle along with a type of data transfer protocol to use; and

The Henseler-Thyagarajan-Kautzleben Combination renders obvious Element [1.c]. EX1003, ¶¶142–145. While the combined teachings of Henseler and Thyagarajan are sufficient to render Element [1.c] obvious, incorporating Kautzleben’s supplemental disclosure regarding a Java-based monitoring system further renders obvious ***notifying a client application manager associated with one or more of the determined grid nodes about adding the new version of the application release bundle***. EX1003, ¶142.

For example, Kautzleben teaches a monitoring system based on the use of MBeans, which are “Java objects that represent [] JMX manageable resources.” EX1007, 4:5–9; EX1003, ¶143. For example, Kautzleben teaches the use of “resource MBeans” (also referred to as “runtime MBeans”), which monitor resources associated with a cluster, and “monitor MBeans,” which are mapped to certain resource MBeans and receive updates regarding the monitored resources. EX1007, 8:30–9:39. The resource MBean may be configured to notify a monitor MBean in various ways, including “push[ing] the monitoring data to the monitor service using a specified event mechanism.” *Id.*, 9:40–45.

As illustrated below in Figure 12, a resource MBean can “be configured to actively transmit monitoring information related to the resource 803 with which they are associated,” through the use of a “**notification** service 1201,” which is triggered by “certain specified events.” EX1007, 11:52–12:19; EX1003, ¶144. For example, after detecting a triggering event in a resource (annotated orange), a resource MBean 625 (annotated yellow) may generate a **notification** to the notification service 1201 (also annotated yellow), which “provides the **notification** to any objects that are registered with the notification service 1201 as listeners,” such as MBean 605 (annotated green). EX1004, 12:1–10. Kautzleben teaches enabling this notification framework by employing a “Notification Broadcaster” class for notifying resource

MBeans and employing a “Notification Listener” class for any objects interested in receiving notifications, such as monitor MBeans. *Id.*, 12:27–36.

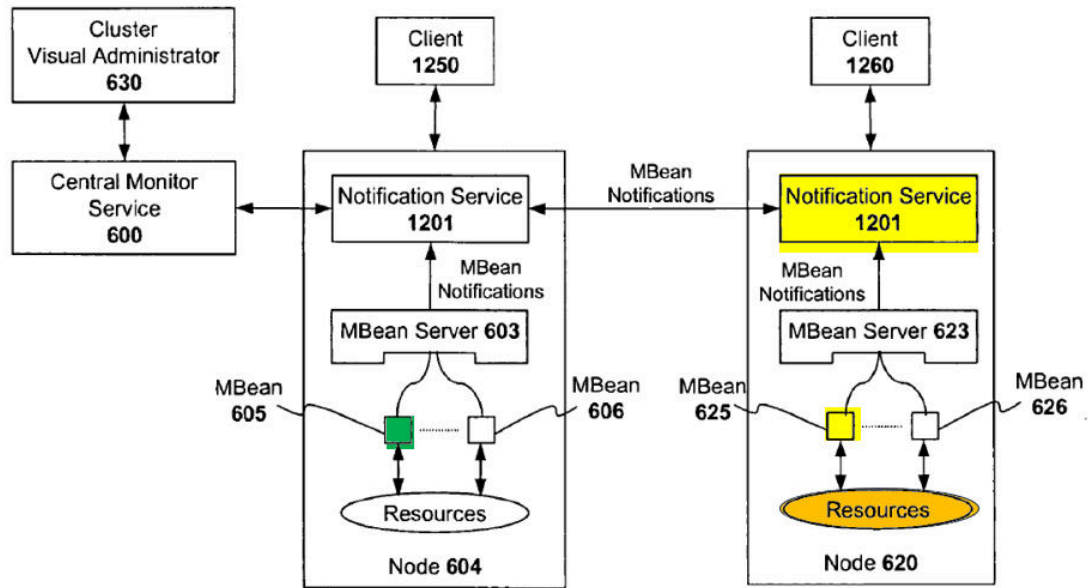


Fig. 12

EX1007, Figure 12 (annotated).

For the reasons explained in Sections V.B.1–3, a POSITA would have been motivated (with a reasonable expectation of success) to incorporate Kautzleben’s teachings into Henseler-Thyagarajan by including a resource MBean and associated notification service into Henseler’s image repository to monitor for and issue a **notification** responsive to a triggering event—the addition of an application image instance (*new version of an application release bundle*, per Ground 1, Element [1.a]). EX1003, ¶145; EX1007, 11:52–12:19. Likewise for the reasons explained in Sections V.B.1–3, and as taught by a Kautzleben, a POSITA would have been

motivated to include a monitor MBean and associated notification service into one or more of Henseler's control node components, including the application governor (*client application manager associated with one or more of the determined grid nodes*, per Ground 1, Element [1.c]), to accept receipt of the issued *notification* from the image repository. EX1003, ¶145.

Element [1.d]: hot deploying/redeploying the new version of the application release bundle on running one or more application servers in an associated grid node using an appropriate hot deployment plug-in based on the data transfer protocol by a respective one of the client application managers.

Supra Ground 1, Element [1.d]. EX1003, ¶146.

(b) **Claim 6**

6. The method of claim 1, further comprising: registering the client application manager associated with one or more of the determined grid nodes with the repository server for accepting notification of receipt of the new version of the application release bundle.

The Henseler-Thyagarajan-Kautzleben Combination renders obvious Claim 6. EX1003, ¶147. To start, as discussed above at Element [1.c], the Henseler-Thyagarajan-Kautzleben Combination features a resource MBean on the image repository (*repository server*) that, via a notification service, issues a notification received by a monitor MBean on the application governor (*client application manager*) upon adding an application image instance (*new version of an application release bundle*). EX1003, ¶147. Moreover, as Kautzleben teaches, the notification

service “provides the notification to any objects that are *registered* with the notification service [] as listeners.” EX1007, 12:1–36; *see also id.*, 8:58–9:39, 11:52–67. In other words, the monitor MBeans, including in the application governor (*client application manager*), would receive the notifications by being mapped to the repository server’s resource MBean and *registering* with the notification service to *accept notifications* from the image repository (*repository server*). *Id.*; EX1003, ¶147.

(c) **Claim 9**

Element [9.pre]: A computer-readable medium with program code embodied therein for causing when executed, a computer system, to perform a method for hot deployment/redeployment in a grid computing environment comprising:

Supra Ground 1, Element [9.pre]. EX1003, ¶148.

Element [9.a]: adding a new version of an application release bundle in a repository server;

Supra Ground 1, Elements [9.a], [1.a]. EX1003, ¶149.

Element [9.b]: determining by a discovery services module which of the one or more grid nodes are running an application associated with the added new version of the application release bundle upon adding the new version of the application release bundle in the repository server;

Supra Ground 1, Elements [9.b], [1.b]. EX1003, ¶150.

Element [9.c]: notifying a client application manager associated with one or more of the determined grid nodes about adding the new version of the application release bundle along with a type of data transfer protocol to use; and

Supra Ground 2, Element [1.c]; Ground 1, Elements [9.c], [1.c]. EX1003,

¶151.

Element [9.d]: hot deploying/redeploying the new version of the application release bundle on running one or more application servers in an associated grid node using an appropriate hot deployment plug-in based on the data transfer protocol by a respective one of the client application managers.

Supra Ground 1, Elements [9.d], [1.d]. EX1003, ¶152.

(d) Claim 12

12. The computer-readable medium of claim 9, wherein the method further comprises: registering the client application manager associated with one or more of the determined grid nodes with the repository server for accepting notification of receipt of the new version of the application release bundle.

Supra Ground 2, Claim 6. EX1003, ¶153.

(e) Claim 14

Element [14.pre]: A computer system comprising: a processing unit; and a memory coupled to the processing unit, the memory having stored therein code for causing, when executed, the processing unit to perform a method comprising:

Supra Ground 1, Element [14.pre]. EX1003, ¶154.

Element [14.a]: adding a new version of an application release bundle in a repository server;

Supra Ground 1, Elements [14.a], [1.a]. EX1003, ¶155.

Element [14.b]: determining by a discovery services module which of the one or more grid nodes are running an application associated with the added new version of the application release bundle upon adding the new version of the application release bundle in the repository server;

Supra Ground 1, Elements [14.b], [1.b]. EX1003, ¶156.

Element [14.c]: notifying a client application manager associated with one or more of the determined grid nodes about adding the new version of the application release bundle along with a type of data transfer protocol to use; and

Supra Ground 2, Element [1.c]; Ground 1, Elements [14.c], [1.c]. EX1003, ¶157.

Element [14.d]: hot deploying/redeploying the new version of the application release bundle on running one or more application servers in an associated grid node using an appropriate hot deployment plug-in based on the data transfer protocol by a respective one of the client application managers.

Supra Ground 1, Elements [14.d], [1.d]. EX1003, ¶158.

(f) Claim 17

17. The system of claim 14, wherein the method further comprises: registering the client application manager associated with one or more of the determined grid nodes with the repository server for accepting notification of receipt of the new version of the application release bundle.

Supra Ground 2, Claim 6. EX1003, ¶159.

VI. PAYMENT OF FEES—37 C.F.R. §42.103

Petitioner authorizes the Patent and Trademark Office to charge any fees to Deposit Account No. 06-1050.

VII. MANDATORY NOTICES—37 C.F.R § 42.8(a)(1)

A. Real Party-In-Interest—37 C.F.R. § 42.8(b)(1)

Petitioners, FedEx Corporation and FedEx Office and Print Services, Inc., are the real parties-in-interest. Additionally, even though Petitioners do not believe that these additional entities are real parties-in-interest, Petitioners further identify Federal Express Corporation and FedEx Dataworks in the interest of completeness.

B. Related Matters—37 C.F.R. § 42.8(b)(2)

Petitioners are not aware of any disclaimers or petitions for inter partes review for the '332 patent. The '332 patent is the subject of a reexamination (reexamination control no. 90/015,497). The '332 patent is also the subject of civil action: *Starbucks Corporation v. Valtrus Innovations Limited et al.*, 2-25-cv-01859 (WDWA) Sep. 25, 2025; *NetApp, Inc. v. Valtrus Innovations Ltd. et al.*, 1-25-cv-06138 (SDNY) July 25, 2025; *Valtrus Innovations Ltd. et al. v. FedEx Corporation et al.*, 2-25-cv-00516 (EDTX), May 9, 2025; *Valtrus Innovations Ltd. et al v. NetApp, Inc.*, 2-25-cv-00517 (EDTX) May 9, 2025; *Valtrus Innovations Ltd. et al v. Lenovo Group Limited et al.*, 2-25-cv-00080 (EDTX), Jan. 27, 2025; *Valtrus Innovations Ltd. et al v. The Home Depot, Inc. et al.*, 2-25-cv-00081 (EDTX), Jan. 27, 2025.

C. Lead And Back-Up Counsel—37 C.F.R. § 42.8(b)(3)

Petitioners provide the following designation of counsel.

Lead Counsel	Backup counsel
Kenneth W. Darby, Jr., Reg. No. 65,068 Fish & Richardson P.C. 60 South Sixth Street, Suite 3200 Minneapolis, MN 55402 Tel: 202-783-5070 Fax: 877-769-7945 Email: IPR39898-0051IP1@fr.com	David Hoffman, Reg. No. 54,174 Fish & Richardson P.C. 60 South Sixth Street, Suite 3200 Minneapolis, MN 55402 Tel: 202-783-5070 Fax: 877-769-7945 Email: IPR39898-0051IP1@fr.com

D. Service Information

Please address all correspondence and service to the address listed above.

Petitioner consents to electronic service by email at IPR39898-0051IP1@fr.com.

Respectfully submitted,

Dated: October 9, 2025

/Kenneth Wayne Darby Jr./
Kenneth W. Darby, Jr., Reg. No. 65,068
David Hoffman, Reg. No. 54,174
Fish & Richardson P.C.
60 South Sixth Street, Suite 3200
Minneapolis, MN 55402
T: 202-783-5070
F: 877-769-7945

(Control No. IPR2026-00039)

Counsel for Petitioners

CERTIFICATION UNDER 37 CFR § 42.24

Under the provisions of 37 CFR § 42.24(d), the undersigned hereby certifies that the word count for the foregoing Petition for *Inter partes* Review totals 13,141 words, which is less than the 14,000 allowed under 37 CFR § 42.24.

Dated: October 9, 2025

/Kenneth Wayne Darby Jr./

Kenneth W. Darby, Jr., Reg. No. 65,068

David Hoffman, Reg. No. 54,174

Fish & Richardson P.C.

60 South Sixth Street, Suite 3200

Minneapolis, MN 55402

T: 202-783-5070

F: 877-769-7945

Counsel for Petitioners

CERTIFICATE OF SERVICE

Pursuant to 37 CFR §§ 42.6(e)(4)(i) *et seq.* and 42.105(b), the undersigned certifies that on October 9, 2025, a complete and entire copy of this Petition for *Inter partes* Review, Power of Attorney, and all supporting exhibits were provided via Federal Express, to the Patent Owner by serving the correspondence address of record as follows:

Botos Churchill IP Law LLP - Malik
430 Mountain Avenue
Suite 401
New Providence, NJ 07974
(908) 738-1770

/Crena Pacheco/
Crena Pacheco
Fish & Richardson P.C.
60 South Sixth Street, Suite 3200
Minneapolis, MN 55402
pacheco@fr.com