Agent based Service Integration for Distributed Problem Solving Environments

Omer F. Rana
Department of Computer Science
University of Wales, Cardiff
POBox 916, Cardiff CF24 3XF, UK
o.f.rana@cs.cf.ac.uk

David W. Walker
Computer Science and Mathematics Division
Oak Ridge National Laboratory
PO Box 2008, Oak Ridge TN 37831-6367, USA
walker@msr.epm.ornl.gov

Abstract

Multi-disciplinary Problem-Solving Environments (M-PSEs) are developed to support sharing of services across multiple application domains. A PSE is, by definition, aimed to support problem solving in a given application domain. However, the infrastructure used to maintain and develop a PSE is not, and various common themes emerge when considering applications across domains. This is the predominant reason for developing M-PSEs, and creating a service layer that can be shared by multiple domain-specific PSEs. An agent-based infrastructure for M-PSEs is described, which enables the integration of legacy codes, specialised visualisation services, numerical libraries and repositories, and resource management systems, such as LSF and Codine. Each "service" in the M-PSE is a dynamic component, that can vary its behaviour based on interactions with other components, or its operating environment.

1 Introduction

Problem Solving Environments (PSEs) can vary in their complexity and size, and range from programs for general scientific/mathematical analysis and visualisation, such as MatLab, Mathematica, and Maple, to large scale distributed environments based on component technologies (CORBA, COM+, Enterprise JavaBeans/Jini) such as the WebFlow/Gateway [1], ADVICE [2], and ARCADE [24] systems. What is often missing from these systems is the ability to integrate these disparate systems, or results generated from them, together under a unified framework. Hence, results generated by MatLab, or simulations performed with ADVICE, cannot be easily shared with other systems, unless a developer modifies file formats or changes definitions used by various systems manually.

What is needed is the ability to combine problem-specific PSEs, facilitating interoperability between various tools and specialised algorithms that each PSE supports. Houstis et al. refer to the infrastructure for developing these as "Multidisciplinary PSEs", which can combine PSEs for tailored, flexible multidisciplinary applications [6]. Based on this general framework, they describe a collection of interacting solver and mediator agents, which can partition large scale problems into a collection of interacting solvers [9, 7]. We extend this notion of collaborating agents to cover code mobility [11], whereby numerical algorithms can be migrated across a network, avoiding the need to migrate large quantities of data. Integration with data sources that are structured, such as object or relational database management systems, are also often missing from scientific software, where the emphasis is generally on parsing files using a custom format specific to the application. Data management is often missing in most high performance computing applications, even though the speed at which data can be moved in and out of secondary or tertiary storage systems is an order of magnitude less than the processing rate. Citing a 1998 NCSA report, Kleese claims that although high performance computers can operate at the TeraFlop range, I/O operations run closer to 10 million bytes per second [13].

A review of what a PSE should contain is first provided, based on requirements identified within other existing projects (currently underway or which have recently completed). A brief overview of these projects is also provided. The agent based infrastructure is then identified, and services that must be supported within such an infrastructure are defined. Two applications are described which make use of this infrastructure. The paper emphasises the importance of agent based services within a distributed PSE, and the novel aspects of this paper are the agent based infrastructure for using services across PSEs, and emphasising the importance of 'knowledge' services which extend beyond the data/syntax level services generally defined in systems such as CORBA and Java (both of which are core implementation tools within existing PSE projects).

Authorized licensed use limited to: Riva Laughlin. Downloaded on November 25,2025 at 23:27:43 UTC from IEEE Xplore. Restrictions apply.
0-7695-0981-9/01 $10.00 (c) 2001 IEEE

2 What should a PSE contain?

A PSE should contain: (1) application development tools that enable an end user to construct new applications, or integrate libraries from existing applications, (2) development tools that enable the execution of the application on a set of resources. According to this definition, a PSE must include resource management tools, in addition to application construction tools, albeit in an integrated way. Component based implementation technologies provide a useful way of achieving this objective, and have been the focus of research in PSE infrastructure (as described in section 2.1). Data management support is also important, especially when using a control or data flow graph to represent an application. In this case data management must be supported for each component in the graph, for the data generated locally at each component. Such support is often missing from existing PSE projects. Based on the types of tools supported within a PSE, we can identify two types of users: (1) application scientists/engineers interested primarily in using the PSE to solve a particular problem (or domain of problems), (2) programmers and software vendors who contribute components to help achieve the objectives of the category (1) users. The PSE infrastructure must support both types of users, and enable integration of third party products, in addition to application specific libraries. Other important services required within a PSE include security for access control and managing software licenses, support for checkpointing component states, support for debugging and exception handling, and component integrity checking.

Existing PSE projects handle these requirements to varying extents. The component paradigm has proven to be a useful abstraction, and has been adopted by many research projects, making use of existing technologies such as CORBA, Enterprise JavaBeans and DCOM/COM. Automatic wrapper generators for legacy codes that can operate at varying degrees of granularity, and can wrap the entire code or subroutines within codes automatically, are still not available. Part of the problem arises from translating data types between different implementation languages (such as complex numbers in Fortran), whereas other problems are related to software engineering support for translating monolithic codes into a class hierarchy. Existing tools such as Fortran to Java translators cannot adequately handle these specialised data types, and are inadequate for translating large application codes. For PSE infrastructure developers, integrating application codes provides one goal, the other being the resource management infrastructure to execute these codes. The second of these can include workstation clusters, or tightly coupled parallel machines. We therefore see a distinction between these two tiers of a PSE, (1) a component composition environment, (2) a resource management system.

A loose coupling between these two aspects of a PSE will be useful where third party resource managers are being used, whereas a strong coupling is essential for computational steering or interactive simulations. A more detailed description can be found in [32].

2.1 Existing PSE Efforts

In this section existing PSE projects, which have become popular and employ some aspects of the infrastructure described previously, are briefly described. The Gateway project [1] introduces a component based system implemented using JavaBeans and utilising dataflow techniques to represent an application as a directed graph. The Gateway system chooses to use the Abstract Task Descriptor (ATD) as its lowest level of granularity of instruction, and to subsequently build up the instructions that define the application. The NCSA "Data to Knowledge" (D2K) [3] project also uses the dataflow approach for integrating components for data mining and knowledge discovery. The Adaptive Distributed Virtual Computing Environment (ADViCE) project [2] is another system that provides a graphical user interface that enables a user to develop distributed applications and specify the computing and communication requirements of each task within the task graph. Unlike the Gateway system, but similar to our own, the ADViCE system has its own scheduler that allocates tasks to resources at run time. The Arcade project [24] uses a slightly different approach in that the system has a three tier architecture, with the first tier consisting of a number of Java Applets that are used individually to specify the tasks (either visually or through a scripting language), to specify resource needs, and to provide monitoring and steering. Each of these Applets then interacts with a CORBA interface which in turn interacts with the final execution user modules distributed over a heterogeneous environment. SCIRun [19] [22] provides a programming environment to support interactive construction, de-bugging, and steering of large-scale scientific applications. The focus in SCIRun is on computational steering, supporting application, algorithm and performance steering. The Distributed Problem Solving Environment Component Architecture Toolkit (CAT) [25] is a component-based toolkit for integrating heterogeneous software components. Aimed specifically at science and engineering, a CAT component can be dynamically inserted into the system and be made to interact

Authorized licensed use limited to: Riva Laughlin. Downloaded on November 25,2025 at 23:27:43 UTC from IEEE Xplore. Restrictions apply.

with other CAT components, regardless of differences between architecture, operating system, and programming language. The end-user interacts with this PSE through a graphical interface, which provides a visual workspace in which components can be created and connected. Before the user can decide which components and machines to employ, she must have access to information about the hardware and software resources available on the system. This facility is provided by the CAT Resource Information Service (RIS). The RIS comprises an "Information Server" which maintains an LDAP database, and stores hardware and software meta-data, and an "Information Browser", a graphical tool packaged with the CAT that allows a user to search and browse the contents of the LDAP database. The Netsolve project [23] enables the user to define problems in a specialised language, not dissimilar to Matlab. Interfaces are also provided for Fortran, C and Java. Netsolve also supports access to both hardware and software based computational resources distributed across a network, supporting load-balancing and resource discovery using a collection of interacting agents. The Parallel ELLPACK project [27] is a PSE for PDE based applications. Implemented using the ELLPACK language and sequential solver libraries, it also contains finite element methods, third party solvers, and a graphical interface for problem specification. Support is also provided for running the generated application on parallel machines.

Other projects which share features of a PSE, but do not provide both a program integration/generation tool and a resource manager include PARDIS [29], PAWS [20], and various resource management systems. Based on existing projects, a PSE must therefore: (1) allow a user to construct domain applications by plugging together independent components. Components may be written in different languages, placed at different locations, or exist on different platforms. Components may be created from scratch, or wrapped from legacy codes; (2) provide a visual application construction environment; (3) support Internet-based task submission; (4) employ an Intelligent Resource Management System to schedule and efficiently run the constructed applications; (5) make good use of industry standards such as middleware (CORBA), document tagging (XML); (6) must be easy for users to extend within their domain. Domain specific additions could be undertaken by application scientists to include new solvers or data sets, or by developers to include new resource or data managers, for instance.

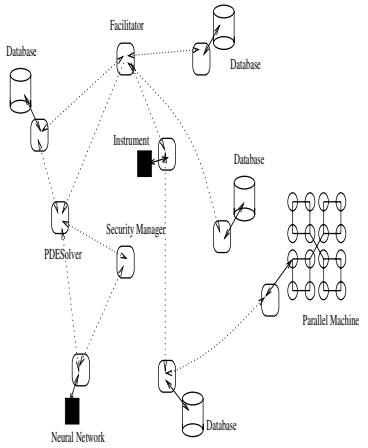

Figure 1: *The 'Agent Grid'*

2.2 Agent based PSEs

Agents can be seen as an extension to objects and components. An agent provides:

- a collection of services (methods in objects),
- behavioural rules (based on offered services) which can change with time and interactions,
- an interaction language,
- a data model (an ontology) that defines a common vocabulary for interactions,
- a strategy or long term goal that the agent intends to pursue.

In this scenario, each agent provides a particular service to other agents, although multiple agents can offer similar services. The offered services or roles for agents can be: (1) resource monitors, (2) match makers, for matching application/task requests with resource capabilities, (3) Partial Differential Equation (PDE) or other numerical solvers, (4) tertiary storage managers, (5) data format converters, (6) security managers, (7) user profilers, etc. These roles are supported by an infrastructure that enables agents to communicate using FIPA ACL [10] – a domain independent agent communication language, and extends the notion of 'Computational Grids' [14], to include a wider range of dynamic information services. Figure 1 illustrates a general 'Agent Grid' (AG) comprised of agents undertaking different roles, wrapping legacy databases, or managing task execution on a parallel machine.

Authorized licensed use limited to: Riva Laughlin. Downloaded on November 25,2025 at 23:27:43 UTC from IEEE Xplore. Restrictions apply.

2.3 Information Grids as Infrastructure for PSEs

Information 'Grids' provide a useful abstraction for connecting and sharing different types of informational resources, which can range from people to application software, and parallel hardware. Generally, there may be a hierarchy of grids, where local grids federate and provide services to a global infrastructure. Local grids in this context may support specialised software libraries or protocols, which are then integrated into some global services. One of the objectives of offering grid services, is to facilitate the pervasive provision of such services, and consequently the automatic discovery of services within a given context. We may infer various kinds of information grids, that could contribute towards multi-disciplinary PSEs, such as:

- *Computational Grids*: These are the most commonly used grid concepts advocated by the high performance computing community, and brought together in the edited work by Foster and Kesselman [14]. Computational grids provide meta-computing toolkits, which range from Globus and Legion for managing resources, high throughput computing infrastructure such as CONDOR, to application-level scheduling mechanisms. This view covers resource description, management, load balancing, data management, leading eventually to the aggregation of computational resources.

- *Geo Grid*: Geographical maps and GPS based infrastructure constitute the Geo Grid, which enables grids to be viewed as cross hatching coordinate systems. Detailed maps provide viewpoints which can range in complexity from building layouts to the entire Earth. Various segments of the Geographic Information Systems community, such as the "Earth Observation Information System" and the "National Image and Mapping Agency", provide and make use of the Geo Grid, for applications in command and control, area demographics, and vegetation and biomass studies.

- *ABIS Information Grid*: The Advanced Battlespace Information System (ABIS) provides a global information grid, as part of the the DARPA I*3, BADD and AICE programs, to connect a large number of data sources to a large number of query sources across the globe. This type of grid is aimed primarily at data integration and management from various different sources, each of which can have a local database schema. An equivalent project in the commercial domain is MCC's InfoSleuth project.

- *Software Grid*: The most widely deployed grid services constitute the "software" grid, which is composed of web servers, email servers and a wide range of other services which can be accessed from geographically dispersed locations. In this context, logical grids operate over a physical infrastructure composed of Ethernet, ATM, Fibre and recently, wireless links. The physical infrastructure is composed of network components, such as routers, gateways, hubs, whereas the logical infrastructure is composed of software services based on distributed objects (CORBA, COM+), Java applets, and a host of other proprietary software and protocols.

- *CoABS Grid*: The DARPA "Control of Agent Based Systems" (CoABS) project [15], is most closely related to our approach, where agents offer various services which can range from component interaction managers, database wrappers, traders/brokers, to resource planners and user interfaces. The scope of the CoABS project is wide, and at present few services are available that may can be integrated effectively with computational or Geo grids. A more detailed account can be found in [16].

Integrating these various types of grids, we can identify some common themes and usage requirements:

- The ability to connect computational resources of differing complexity, to improve resource utilisation, and enable pervasive access to these resources. Computational resources and applications may dynamically enter or leave the grid, and be offered at various levels of granularity. Resources can include computational and visualisation engines, data repositories, and scientific instruments.

- Connect geographically dispersed users to software and hardware resources in a transparent way, whereby users can perform complex mathematical operations remotely. As a consequence of performing these mathematical operations, to also manage storage resources in an efficient way, to divide the recorded data between tapes and disks in a hierarchical manner, and subject to some efficiency criteria.

- Connect to newer and legacy systems simultaneously, and undertake format conversions between data stored in file systems at geographically distributed sites.

Authorized licensed use limited to: Riva Laughlin. Downloaded on November 25,2025 at 23:27:43 UTC from IEEE Xplore. Restrictions apply.

- Make use of existing numerical and scientific software, in libraries such as the Guide to Available Mathematical Software (GAMS) [4], and execute the software on-demand, at the point of data availability.

- Configure and re-organise software components to create applications dynamically, where components are self-identifying, contain constraints on their licensing needs, and are self-documenting to identify their particular features.

PSE projects must therefore make use of grid services where possible, rather than create their own versions of these. This is particularly important if resources and data sets need to be shared between users, or if multi-disciplinary research needs to be undertaken where data sets from prior experiments need to be further analysed. Data fusion is becoming increasingly important in the context of scientific applications, where data gathered by different instruments, or generated from multiple experiments, must be integrated.

3 Service Integration for Multi-Disciplinary PSEs

We suggest the extension of the component model to agents, where an agent can contain behaviour rules, and interacts with other agents using a specialised communication language. From this viewpoint, agents become both application generators and managers, and resource managers that must execute these applications. Each agent may make use of grid based services, such as PDE solvers, which are also implemented as agents. All interactions are therefore either requests for services, or responses carrying results. Users interact with a presentation agent, which is responsible for generating an application on a visual canvas. Once completed, the application is subsequently analysed for errors or omissions by another agent, which interact with agents providing grid services to find suitable solvers or data sources User agents can cater for the two category of users identified previously. Hence, agents for application developers can facilitate the 'checking in' of components into repositories, or ensuring that the provided service adheres to a common data model. Similarly, agents for application scientists can help users locate services of interest, locate data sets of interest, or translate data formats to be usable by particular solvers. Where multiple services are offered, a market (auction) protocol, such as the Contract Net or WALRAS [17] protocol, may be used to resolve the conflict within a given number of message exchanges.

The following services need to be provided, in order for the agent based approach to work:

- A domain vocabulary to support semantic interoperability between interacting agents. Each agent must utilise terms that another agent can understand, and this will be aided by the development of specialised ontologies for scientific computing. These ontologies can vary in granularity, from representing specialised terms within a given scientific domain (such as Molecular Dynamics), to common terms which may be applicable across a wide range of domains, such as matrix solvers, PDE solvers, for instance. In cases where a domain specific term cannot be resolved, the participating agents should automatically revert to a domain independent ontology, closest to the domain specific one. Domain ontologies may be encoded in XML, for instance, and be carried as a reference within each message that is communicated between agents.

- Identification of services which are domain independent and those that are domain specific, and correspondingly the identification of specialised roles that need to be undertaken within PSEs. These roles can vary in granularity and complexity, and must be described using the common ontological terms, to enable utilisation by applications and by other grid services.

- Wrap computational and data resources as agent services, describing resource capabilities and application requirements. The class advertisement approach adopted in CONDOR provides a useful way of achieving this objective, however the ontological scheme should be consistent.

- Associate a goal with each resource, based on the role undertaken by the resource. The goal can be to improve utilisation over a given time frame for a computational resource, or it may be to complete specific task within a given time. The use of goals will enable each resource to operate in a de-centralised manner, although goals of multiple resources (such as in a cluster) may be combined. Goals reflect management policies for a given type of software or hardware resource, and can also vary with time, based on a change in the environment within which the resource operates.

4 Applications

We describe two applications which make use of the agent infrastructure previously defined. An image processing application which supports on-demand process-

Authorized licensed use limited to: Riva Laughlin. Downloaded on November 25,2025 at 23:27:43 UTC from IEEE Xplore. Restrictions apply.

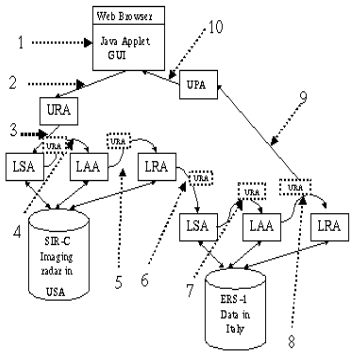

Figure 2: *Agent based system for the Synthetic Aperture Radar Atlas. See section 4.1 for abbreviations and details.*

ing of images, and a resource management system which can deal with computational and storage devices.

4.1 Synthetic Aperture Radar Atlas (SARA)

We first describe an application based on a set of collaborating agents for processing images generated by Synthetic Aperture Radar. The SARA images are generated by transmitting electromagnetic pulses to the surface of the Earth at three wavelengths (L-band (23.5cm), C-band (5.8cm) and X-band (2.5cm)), and measuring the corresponding backscatter, which is subsequently collected and analysed. This approach makes use of a Geo Grid and aspects of Computational Grids. We combine these services to propose an Agent Grid, where agents undertake services generally offered by disparate systems, such as a BeoWulf cluster, an HPSS storage system, a Geographic Information System, and a web interface for presentation to the user.

In the SARA system [8], the user selects a region of the Earth by drawing a polygon to identify the region of interest on a map. This request is then submitted to a metadata server, where the latitude and longitude values for the positions identified by the user, are converted into files names on the HPSS/Unitree system. The retrieved image is then analysed using approaches ranging from principle component analysis to neural networks. In the proposed system, we wrap all of these resources as collaborating agents, as illustrated in figure 2. Hence, a User Request Agent (URA) is generated from the user interface, and carries an analysis algorithm to the data source. The URA represents a mobile agent, which enables the migration of an analysis algorithm to the point of data. The URA interacts with a Local Security Agent (LSA) at the remote site, which authenticates the incoming agent, and provides a shell within which the URA can operate. This shell determines security levels that must not be violated by the incoming URA, and if they are, the URA is terminated. Once the check has been completed, the operation requested by the URA is examined by the Local Assistant Agent (LAA), to ensure that the required data source is on-line and accessible. On successful verification, the URA connects to a Local Request Agent (LRA) which interacts with the local data store to execute the analysis algorithm on the image. The LRA hides the complexity of retrieving data from the secondary/tertiary storage, and schedules disk and tape requests to storage media under its control. Once the required operation has been performed, the URA can either take the processed image back to the user, where a User Presentation Agent (UPA) can display it to the user, or the URA can migrate to another site, containing a Geographic Information System to overlay the processed image with geographic features such as towns and roads.

In this case, each agent undertakes a particular role, and is responsible for achieving the role within a given time. Each agent therefore has a goal function that is connected with its role, and each agent offers a service within a given context. In the SARA system, agents communicate through specialised messages, which carry a role specific ontology, and a common domain ontology. For instance, the LRA must be able to receive a request to process an image, the LAA must be able to interpret a request to check the availability of a data source, and the LSA must be able to check the security certificate of the incoming URA. Interactions between agents use tagged messages, which are divided into a "performative" and a "content" part. A "performative" identifies a particular operation that the recipient agent must undertake, and are defined by the ACL standard [10]. The "content" portion contains variable declarations, identifying constraints or a reference to a common domain ontology that the communicating agents must use. For instance, the URA and the LRA interact as:

```
(ask-one,
 sender: URA:o.f.rana:131.251.42.111,
 receiver: LSA:saraserv:131.215.49.4:8755,
 in-reply-to: ID,
 content: (BAYES ?image,
  source: BAYES CART,
  language: Java,
  ontology: href:131.215.49.4/analysis.xml)
 ontology: sara-XSIL,
 language: Java
)
```

where the URA requests the LRA to perform a particular query on its behalf, on the data source contain-

Authorized licensed use limited to: Riva Laughlin. Downloaded on November 25,2025 at 23:27:43 UTC from IEEE Xplore. Restrictions apply.

Ex.1136 / IPR2025-00915 / Page 6 of 10
Intel Corporation v. Advanced Cluster Systems, Inc.

ing the images. The `ask-one` performative indicates that the URA only requires one response from the LSA, in case multiple matches are found. Other performatives can include `ask-all`, or `tell` (where no response is required). The content field constrains a variable `image`, which must be analysed by a Bayesian CART algorithm, which is carried with the URA, written in Java, and adheres to a domain ontology `analysis.xml`, for analysis algorithms. The `href` tag identifies where the particular ontology can be found. There is also a domain specific ontology for the particular application within which this analysis algorithm is being used, which is the `sara-XSIL` ontology. Hence, the URA can delegate responsibility for performing the Bayesian analysis to the LSA, and expects the LSA to send back an image on completion. All other agent interactions are similarly defined.

4.2 Resource Management and Discovery

A second application is a decentralised resource management system that makes use of 'resource capabilities' and 'task requirements' to find suitable allocations. Our system can deal with a dynamically changing environment, which could include new tasks created at run time, or new devices added to existing clusters. The approach makes use of various algorithms that use similarities between devices to find approximate matches of tasks to resources. Each resource administrator is required to describe the resource using a policy description scheme, which is subsequently used by a MatchMaker agent to find suitable allocations. Resource selection or discovery generally involves identifying suitable computational engines from a pool, mostly homogeneous, based on criteria ranging from licensing constraints to processor(s) capability(ies) and background workload. In task-parallel programs, different tasks may need to be mapped to different resources, whereas in the data parallel case, data decomposition becomes significant. Existing resource management systems, such as the Load Sharing Facility (LSF), for instance, involve a queueing facility to which application tasks are submitted. Such systems are primarily aimed at managing a homogeneous cluster, rather than a heterogeneous resource pool. In addition, the process of identifying a suitable queue to which tasks must be submitted is delegated to the user (either directly or via a job control language). The proposed approach uses the object-oriented description mechanism in Legion [26], but is most closely related to the 'class advertisement' mechanism in CONDOR [31].

The following steps show how the MatchMaking service operates. R_j is an arbitrary resource manager; M is the centralised MatchMaker; D_R is a resource document; D_T is a task document. An arbitrary task is defined as T_i. Hence:

1. Each R_j sends an asynchronous message to a predefined MatchMaking service 'M' (running on a host with a fixed IP address) to indicate its availability within a cluster. Each message is tagged with the resource type: (1) computational resource 'C', (2) data storage resource 'S', (3) visualisation resource 'V', or (4) scientific instrument 'I'. For compound resources which can be of multiple types, the letters can be aggregated.

2. On receiving the message, the local 'M' responds by sending a document specifying the required information to be completed by the resource manager at R_j. This information is encoded in an XML document, and contains specialised keywords that correspond to dynamic information that must be recorded for every device in the pool. The form also contains a time stamp indicating when it was issued, and an IP address for the MatchMaking service. The form can either be automatically completed using agents running on the resource (similar to daemon processes, but aimed at interacting with the MatchMaker), or it can be completed manually by a systems administrator.

3. The manager for R_j completes the document, and sends it back to 'M', maintaining a local copy. The document contains the original time stamp of 'M', and a new time stamp generated by R_j. Some parts of the document are static, while others can be dynamically updated. Once this has been achieved, the new device is now registered with the resource manager, and will continue to be a suitable candidate for task allocation until it de-registers with 'M'. If a device comes off-line or crashes, 'M' will automatically de-register it when it tries to retrieve a new copy of the document. We define each resource document as D_R.

4. Similarly, a user wishing to execute an application uses a request document based on requirements of each task T_i within an application, and classed using either 'C', 'S', 'V' or 'I' annotation to the MatchMaking service 'M' within the local cluster. This results in a set of documents being sent to the user, for each T_i in the application. The user has complete control over the granularity of T_i, and tasks may be grouped based on known dependencies. Each document must now be completed by the user, either using pre-defined scripts or manually. This issued document contains a time stamp, and on subsequent return to 'M', contains a time

Authorized licensed use limited to: Riva Laughlin. Downloaded on November 25,2025 at 23:27:43 UTC from IEEE Xplore. Restrictions apply.

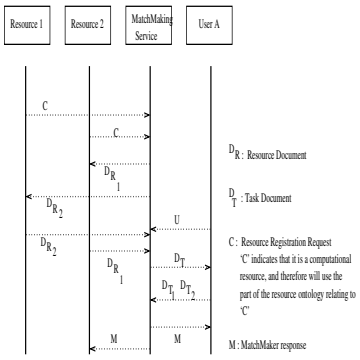

Figure 3: *Sequence diagram between MatchMaker and other services*

stamp from the user. We define such a document as D_T.

5. 'M' now tries to find a match between each D_T and D_R based on a pre-defined match making criteria. This criteria can include a direct syntax match between the keywords in the two documents, or a classification ontology maintained by the MatchMaker. The ontology defines device capability and task requirements, based on keywords in the document (as defined below). Each time a suitable match is made, 'M' sends the generator of D_T and D_R their corresponding identities. The matched participants must now activate a separate protocol to complete the allocation, and this process does not involve 'M'. The matched R_j must now de-register itself, or request a new D_R.

6. If a local 'M' within a cluster cannot fulfil a request based on the submitted D_R from resources to which it is connected, it can forward this request to an 'M' within another cluster. The MatchMaking services are therefore federated, and register with each other using a pre-defined document D_M, which identifies their IP address and start time.

The interactions between various participants in the resource management system are illustrated in the sequence diagram of figure 3. For instance, when a resource agent sends a message to the MatchMaker, the later should be able to look at it's acquaintance table (stored advertise messages D_R) to retrieve the content, which is an `achieve` performative, and would be sent back to the resource agent. This would cause the resource agent to complete its D_R (as it exists at that time) and return this to 'M'. If multiple messages need to be exchanged between 'M' and the resource agent, a `reply-with` and a `in-reply-to` performative

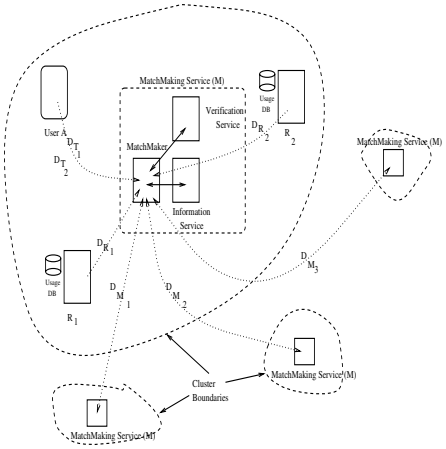

Figure 4: *MatchMaking architecture*

is used to keep track of the `conversations`. In this case, a `conversation` object in JKQML (an implementation of KQML in Java, available from IBM [5]) connects exchanged messages with `conversation` identifiers. Hence, a resource agent would send a message:

```
(advertise
  :sender parian.cs.cf.ac.uk:8100
  :receiver url://parian.cs.cf.ac.uk:20001
  :reply-with km.getInitialID()
  :language ACL
  :ontology resource
  :content  ( (achieve
              :sender parian.cs.cf.ac.uk:20001
              :receiver parian.cs.cf.ac.uk:8100
              :reply-with km.getInitialID()
              :language ACL
              :ontology resource
              :content ( ) ) ) )
```

A message exchange between a resource agent and 'M'

Figure 4 illustrates the MatchMaking architecture, comprising of a single MatchMaker within a cluster. 'M' consists of three core components, (1) an information service, (2) a verification service, (3) the matchmaking service itself. The information service is responsible for obtaining dynamic parameter values within documents, and can interact with a local resource manager or a user to obtain these parameters. At any given time, the final version of a document is always maintained with the MatchMaker, and the information service merely acts to facilitate the gathering process. The verification service is used to check information maintained on a given resource by invoking the information service, and is used to check submitted documents to ensure that all necessary information has been supplied. As illustrated, a user or resource can

Authorized licensed use limited to: Riva Laughlin. Downloaded on November 25,2025 at 23:27:43 UTC from IEEE Xplore. Restrictions apply.

submit multiple documents, corresponding to differing granularities within the application or the resource.

The verification service contains an XML parser and a Document Type Definition (DTD) for D_R, D_T and D_M. All submitted documents can be verified prior to processing by the MatchMaker. Further, modifications may be made to any of these documents, and the next resource to register will get the newer version of the document, with a change in the document version number sent to the resource or task.

Each resource can also store a usage history in a local database, based on the D_R schema, at intervals determined by the resource administrator. In addition, based on a local policy, an Administrator may refuse to record certain parameters within D_R. MatchMakers in different domains/clusters can interact with each other, and only do so if locally generated tasks cannot be executed on local resources. The usage history database can also be used to maintain composite metrics, such as load averages over a given time period. The resource manager can either extend D_R with additional tags, or query the MatchMaker to supply a more detailed document for completion. D_R, on which the resource ontology is also based, is defined as:

```
<resource>
 <name value="parian.cs.cf.ac.uk"
     short="parian.cardiff">131.251.42.5</name>
 <type compound="true" value="0">C</type>
 <type compound="true" value="1">S</type>
 <operatingsys>Solaris7</operatingsys>
 <arch>SUN Ultra</arch>
 <loadavg type="DYNAMIC">0.332</loadavg>
 <idletime type="DYNAMIC" value="seconds">1442</idletime>
 <processcount type="DYNAMIC" value="NULL">120</processcount>
 <memfree type="DYNAMIC" value="MB">64</memfree>
 <memory type="cache" value="MB">4</memory>
 <memory type="RAM" value="MB">128</memory>
 <storage type="disk" value="GB">8</storage>
 <mmtimestamp>01.01.2000.15.15</mmtimestamp>
 <submittimestamp>02.01.2000.23.22</submittimestamp>
 <docversion>1.0</docversion>
 <permission value="allow">cs.cf.ac.uk</permission>
 <permission value="allow">doc.ic.ac.uk</permission>
 <permission value="prevent">ecs.soton.ac.uk</permission>
 <constraint type="KIF">
   <memfree value="MB"> gt 64</memfree>  &&
   <idletime value="seconds"> gt 1000 </idletime>
 </constraint>
</resource>
```

The system has been implemented using JKQML. Figure 5 illustrates the resource document The system has been demonstrated for matching computational libraries running on particular workstations, with tasks which have been dynamically created from a user interface.

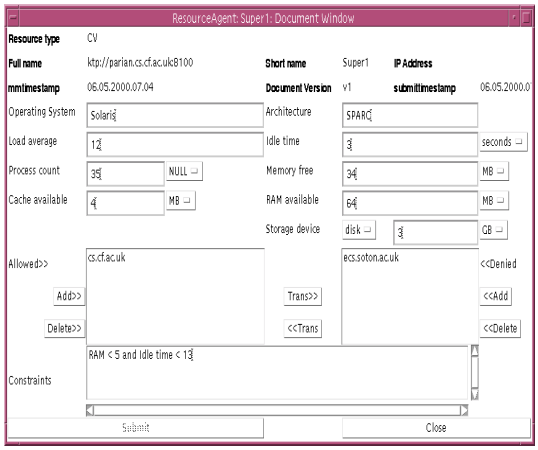

Figure 5: *Interface for resource agent – completing the resource documented* D_R

5 Conclusion

We propose an agent based integration of services needed to create a multi-disciplinary PSE infrastructure. Agents enable computational, data and application resources to be viewed in a unified way, enabling the abstraction of implementation details from application scientists and engineers. We identify common services that must be provided to enable such an infrastructure to operate, and use two applications based on collaborating agents to demonstrate the concepts.

The agent based approach provides the most cost effective way to bring together people, software and hardware resources to construct PSEs. This is due to the use of an "agent" abstraction, that can be universally applied within the system, for software and hardware resources and for user support. Existing numerical and scientific software can be wrapped as an agent, and run in its native environment. Hence, no re-writes are necessary for existing codes. The agent wrapper provides a interaction layer to communicate with other agents, and execution support rules which modify when and how access to the scientific code is achieved. In this case, wraping is not to translate between data types in different programming language (converting Fortran to Java, for instance), but for adding aditional functinality to an existing code. The communication layer can trigger execution of the original numeric or scientific code, giving no loss in performance. Application scientists can then make use of various numerical solvers, computational hardware, data sources, visualisation tools, and access these services via portals that may be fixed or mobile.

Authorized licensed use limited to: Riva Laughlin. Downloaded on November 25,2025 at 23:27:43 UTC from IEEE Xplore. Restrictions apply.

References

[1] Geoffrey Fox, Tomasz Haupt, Erol Akarsu, Alexey Kalinichenko, Kang-Seok Kim, Praveen Sheethalnath, and Choon-Han Youn, *The Gateway System: Uniform Web Based Access to Remote Resources*, Proceedings of ACM JavaGrande Conference, San Francisco CA, June 1999

[2] Kok K. Kee and Salim Hariri, *The Software Architecture of a Virtual Distributed Computing Environment*, Proceedings of HPDC, Portland, Oregon, 1997

[3] D2K: Environment for Data Mining, see Web site at: `http://chili.ncsa.uiuc.edu`.

[4] Guide to Available Mathematical Software, see Web site at: `http://gams.nist.gov/`.

[5] IBM Research. Implementation of the Knowledge Query and Manipulation Language in Java. See Web site at: `http://www.alphaworks.ibm.com`.

[6] E. N. Houstis, A. Joshi, J. R. Rice, T. Drashansky, and S. Weerawarana, *Towards Multidisciplinary Problem Solving Environments*, HPCU News, Department of Computer Science, Purdue University, W. Lafayette, IN 47907-1398, USA, 1998

[7] T. Drashanasky, E. N. Houstis, N. Ramakrishnan, and J. R. Rice, *Networked Agents for Scientific Computing*, Communications of the ACM., vol. 42, no. 3, pp. 48-54, March 1999.

[8] Roy Williams *The Synthetic Aperture Radar Atlas*, See web site at: `http://www.cacr.caltech.edu/sara/`

[9] P. Tsompanopoulou, L. Boloni, D.Marinescu and J. Rice, *The Design of Software Agents for a Network of PDE Solvers*, Proceedings of workshop on Agent based High Performance Computing, at third annual conference on Autonomous Agents, Seattle WA, May 1999

[10] Foundation for Intelligent Physical Agents (FIPA), *Agent Communication Languages – Spec 2, Draft, version 0.1*, See web site at: `http://www.fipa.org/`, 1999

[11] Omer F. Rana and David W. Walker, *Bringing together Mobile Agents and Data Analysis in PSEs*, PDPTA99, Las Vegas, June 1999

[12] Craig Thompson, Tom Bannon, Paul Pazandak and Venu Vasudevan, *Agents for the Masses*, Proceedings of workshop on Agent based High Performance Computing, at third annual conference on Autonomous Agents, Seattle WA, May 1999

[13] Kerstin Kleese, *High Performance Computing Needs High Performance Data Management*, Technical Report, High Performance Computing Group, CLRC – Daresbury Laboratory, Daresbury, Warrington, Cheshire WA4 4AD, UK, 2000

[14] I. Foster and C. Kesselman (eds.), *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufman, 1998

[15] DARPA Project, *CoABS: Control of Agent Based Systems*, See web site at: `http://coabs.globalinfotek.com/`, 2000

[16] Frank Manola, *Characterizing Computer-Related Grid Concepts*, Object Services and Consulting, Inc. See web site at: `http://www.objs.com`, December 29, 1998

[17] J. Q. Cheng and M. P. Wellman, *The WALRAS algorithm: A convergent distributed implementation of general equilibrium outcomes*, Computational Economics, 12, 1998

[18] The JavaGrande Forum. See web site at: `http://www.javagrande.org/`.

[19] SCIRun: Scientific Computing and Imaging. See web site at: `http://www.cs.utah.edu/ sci/`.

[20] Peter Beckman, Patricia K. Fasel, William F. Humphrey, and Susan M. Mniszewski. Efficient Coupling of Parallel Applications Using PAWS. *Proceedings of High Performance Distributed Computing (HPDC) 7 Conference, Chicago*, 1998.

[21] R. Bramley and D. Gannon. PSEWare. See web site at: `http://www.extreme.indiana.edu/pseware`.

[22] C. Hansen G. Kindlmann C. Johnson, S. Parker and Y. Livnat. Interactive Simulation and Visualization. *IEEE Computer*, December 1999.

[23] Henri Casanova and Jack Dongarra. NetSolve: A Network Server for Solving Computational Science Problems. *International Journal of Supercomputer Applications and High Performance Computing*, 11(3):212–223, 1997.

[24] Zhikai Chen, Kurt Maly, Piyush Mehrotra, and Mohammad Zubair. Arcade: A Web-Java Based Framework for Distributed Computing. See web site at: `http://www.icase.edu:8080/`.

[25] Dennis Gannon and Randy Bramley. Component Architecture Toolkit. See web site at: `http://www.extreme.indiana.edu/cat/`.

[26] A. S. Grimshaw. Campus-Wide Computing: Early Results Using Legion at the University of Virginia. *Int. Journal of Supercomputing Applications*, 11(2), 1997.

[27] E. N. Houstis, J. R. Rice, S. Weerawarana, A. C. Catlin, P. Papachiou, K.-Y. Wang, and M. Gaitatzes. Parallel ELLPACK: A Problem Solving Environment for PDE Based Applications on Multicomputer Platforms. *See web site at:* `http://www.cs.purdue.edu/research/cse/`.

[28] D. R. Jones, D. K. Gracio, H. Taylor, T. L. Keller, and K. L. Schuchardt. Extensible Computational Chemistry Environment (ECCE) Data-Centered Framework for Scientific Research. *in Domain-Specific Application Frameworks: Manufacturing, Networking, Distributed Systems, and Software Development, Chapter 24, Published by Wiley*, 1999.

[29] Katarzyna Keahey and Dennis Gannon. PARDIS: CORBA-based Architecture for Application-Level PARallel DISTributed Computation. *Proceedings of Supercomputing97*, November 1998.

[30] Vijay Menon and Anne E. Trefethen. MultiMATLAB: Integrating MATLAB with High-Performance Parallel Computing. *Proceedings of SuperComputing97*, 1997.

[31] Raman R, M. Livny, and M. Solomon. Matchmaking: Distributed Resource Management for High Throughput Computing. *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing*, july 1998.

[32] D. Walker, M. Li, O. Rana, M. Shields, and Y. Huang. The Software Architecture of a Distributed Problem Solving Environment. Technical report, Oak Ridge National Laboratory, Computer Science and Mathematics Division, PO Box 2008, Oak Ridge, TN 37831, USA, December 1999. Research report no. ORNL/TM-1999/321.

Authorized licensed use limited to: Riva Laughlin. Downloaded on November 25,2025 at 23:27:43 UTC from IEEE Xplore. Restrictions apply.