Scalable Computing: Practice and Experience
Volume 6, Number 4, pp. 83–94. http://www.scpe.org
ISSN 1895-1767
© 2005 SWPS

AGENT BASED SEMANTIC GRIDS: RESEARCH ISSUES AND CHALLENGES

OMER F. RANA* AND LINE POUCHARD†

Abstract. The use of agent based services in a Computational Grid is outlined—along with particular roles that these agents undertake. Reasons why agents provide the most natural abstraction for managing and supporting Grid services is also discussed. Agent services are divided into two broad categories: (1) infrastructure services, and (2) application services. Infrastructure services are provided by existing Grid management systems, such as Globus and Legion, and application services by intelligent agents. Usage scenarios are provided to demonstrate the concepts involved.

1. Introduction and Related Work. There has been an increase in interest recently within the Grid community [11] towards "Service Oriented" Computing. Services are often seen as a natural progression from component based software development [6], and as a means to integrate different component development frameworks. A service in this context may be defined as a behaviour that is provided by a component for use by any other component based on a network-addressable interface contract (generally identifying some capability provided by the service). A service stresses interoperability and may be dynamically discovered and used. According to [7], the service abstraction may be used to specify access to computational resources, storage resources, and networks in a unified way. How the actual service is implemented is hidden from the user through the service interface. Hence, a compute service may be implemented on a single or multi-processor machine—however, these details may not be directly exposed in the service contract. The granularity of a service can vary—and a service can be hosted on a single machine, or it may be distributed. The "TeraGrid" project [9] provides an example of the use of services for managing access to computational and data resources. In this project, a computational cluster of IA-64 machines may be viewed as a compute service, for instance—hiding details of the underlying operating system and network. A developer would interact with such a system using the GT4.0 [26] system—via a collection of services and software libraries.

Web Services provide an important instantiation of the Services paradigm, and comprise infrastructure for specifying service properties (in XML—via the Web Services Description Language (WSDL) for instance), interaction between services (via SOAP), mechanisms for service invocation through a variety of protocols and messaging systems (via the Web Services Invocation Framework), support for a services registry (via UDDI), tunnelling through firewalls (via a Web Services Gateway), and scheduling (via the Web Services Choreography Language). A variety of languages and support infrastructure for Web Services has appeared in recent months—although some of these are still specifications at this stage with no supporting implementation. Web Services play an important role in the Semantic Web [17] vision, aiming to add "machine-processable information to the largely human-language content currently on the Web" [12]. A list of publicly accessible Web Services (defined in WSDL) can be found at [21]. By providing metadata to enable machine processing of information, the Semantic Web provides a useful mechanism to enable automatic interaction between software—thereby also providing a useful environment for agent systems to interact [8]. The adoption of more complex representation schemes for metadata, such as WebONT [13], suggest that the software using this information can be more adaptive, and support updates when new information becomes available. The agent paradigm therefore provides a useful mechanism for managing and mediating access to Web Services. Various extensions of Web services through the agents paradigm have been discussed by Huhns [8]—the most significant in the context of Grid computing include self-awareness and learning capability, the ability to support a number of ontologies, and the formation of groups or teams of agents. Conversely, a key advantage of using agents is to support semantic interoperability (i. e. interaction between software systems based on pre-agreed, semantically grounded, definitions). Support of technologies such as WebONT in the context of Web Services are likely to provide the necessary core infrastructure for agents to work more effectively in dynamic environments such as Computational Grids.

2. Role of Agents in Grids. Grid computing currently focuses on sharing resources at regional and national centres. Generally, these include large computational engines or data repositories, often requiring the user to accept "usage policy" statements from the centre managers and owners. Similarly, resource owners are

[*] School of Computer Science, Cardiff University, UKo.f.rana@cs.cf.ac.uk

[†] Computer Science and Mathematics Division, Oak Ridge National Laboratory, PO Box 2008, Oak Ridge TN 37831-6367, USA pouchardlc@ornl.gov

obliged under the policy to guarantee access once an external user has been approved. Access rights to the resources are supported through X.509 certificates—whereby a user requiring access must posses a certificate. The `grid-proxy-init` function in Globus provides a mechanism for delegation—however, it is limited in scope, and protected mainly by standard Unix access rights. In this model, a trust-chain must be established before a proxy request can be accepted. Furthermore, system administrators responsible for particular resource domains are accountable—and operate based on the policy of the site. As Grid systems embrace service-oriented computing, more open and flexible mechanisms are necessary to support service provision and service usage, as a user providing a service may not belong to a particular centre. Hence, multiple providers may offer a similar service, and the service user now has to select between them. The more "open" perspective on Grids—whereby service providers can be a collection of centres or individuals—would necessitate a user evaluating service providers based on a number of different criteria, such as: choosing services which are best value for money, choosing the most "reputable" services, choosing the most secure services, or services which have the highest response (execution) time, or which have been around the longest. These criteria are therefore more diverse in scope, and can support service choice based on dynamic, run-time attributes of a service. We assume two kinds of services to exist within a Grid: (1) core services—which are provided by the infrastructure and by trusted users, and (2) user services—which can be provided by any participant utilising common Grid software—such as OGSA. Two such core services—responsible for managing access to user services—include:

- Certificate Authority (Security Service): The certificate authority is externally managed, and used to authenticate services—based on the identity of a service provider. Only a few of these services are likely to exist across a Grid—and aimed at ensuring that service providers can be verified. The Certificate Authority services is also used to support the development of service contracts between a service user and provider. A simple mechanism based on X.509 certificates already exists, and additional work is necessary to extend this to include users who require temporary certificates, or may change their identity over time. A criteria to be associated with such a service includes the "risk" of accessing a service which does not posses a certificate. In this context, the service user must now decide whether to not accept any service at all, or to choose one which is non-trustable. Such risk evaluation must be undertaken with other decisions being made by the service user—and within a limited time. The decision making capability needed to undertake such an evaluation can be supported through agent systems—and has been a subject of extensive research as "trust models" [31]. The concept of risk can be defined in a number of different ways—for instance, a high risk service may be one that is likely to give low-accuracy results (for a numeric service), or one that is provided by an unknown vendor. It is therefore important to qualify what is meant by risk in a particular instance.
- Reputation Service: Each service can have an associated "Reputation" index, which is used to classify how often the provider has fulfilled its Service Level Agreement (contract) in the past, and to what degree of confidence. It is possible for a particular service user to subscribe to multiple such Reputation Services—and indeed for a client service to look up the reputation of the providing service from multiple Reputation providers. The concept of Reputation Services have been developed in the Peer-2-Peer computing community [14], and aimed at increasing accountability within a system of anonymous peers. Another concept of reputation (in the FreeHaven project [15]) requires service owners to provide "receipts" (feedback) to verify the correctness of results obtained from other services they interact with. These receipts are coupled with services that act as "witnesses" to ensure that receipts have been generated, and thereby can judge node misbehaviour. In the context of Grid services, witnesses can be external nodes which monitor that a given node has met its Service Level Agreement, and can verify that the feedback provided by the user on the service provider is accurate.

A Reputation or Certificate can be used by a client service to identify whether to use a particular service provider. This confidence in a given service is important in the context of service-oriented Grids—as it allows requesting services to select between multiple providers with a greater degree of accuracy. Agents provide the most suitable mechanism for offering and managing Grid services. Each agent can be a service provider or user, or can interact with an existing information service.

We therefore assume that services within a Grid environment are managed and executed via agents. It is also possible for each agent to support one or more "service types" (see section 4.2). We assume three kinds of agents to be present: (1) Service Providers, (2) Service Consumers, and (3) Community Managers (see section 4.1). Each agent must therefore provide support for managing a community description, managing and sustaining interactions with other agents, and provide a policy interpreter. The policy interpreter is used by a service provider and a community manager to ensure that a service provider conforms to its service provision contract. Particularly important in Grid systems is the role played by middle agents—primarily service providers which do not offer an application service, but act as brokers to discover other services of interest. The criteria for service discovery used by a broker may range from service type to service reputation—and a service consumer may simultaneously invoke a number of different service providers (brokers) to undertake this search.

The particular challenges therefore include the ability to assess the risk associated with using a service, and provide feedback to potential users to evaluate this risk. Middle agents can support the management of risk within an agent community—enabling agents to combine the use of trusted services along with newer ones.

3. Service Lifecycle. Each agent is responsible for managing one or more services—and each agent may utilise a number of different infrastructure services to achieve this. An agent exists within a particular community, and utilises infrastructure services (such as a security or registration service) within its community first. A service lifecyle identifies the stages in creating, managing, and terminating a service. A new service may either be created by an agent, or a service may be associated with an agent by a user—where access to the service is subsequently mediated by the agent. A new service may also be created by combining services offered by different agents—whereby an agent manages a service aggregate. The agent is now responsible for invoking services in the order specified in the composition process (specified in a service enactment contract). Once a new service has been created, it must be registered with its "community manager" by the agent. A service is initialised and invoked by sending a request to the agent managing the service, which may either agree to the request immediately, or offer a commitment to perform the service at a later time. Service termination involves an agent unregistering a service via the community manager, and removing all data corresponding to the service state. When an agent needs to execute an aggregate service, it will involve interactions with agents within multiple communities. The manager within each community is responsible for ensuring that service contracts are being adhered to by agents within its community. The ability to create a service aggregate leads to the formation of "dynamic workflow"—whereby an agent decides at run time which other agent it needs to interact with to achieve a particular goal. Consequently, the exact invocation sequence between services is not pre-defined, and may vary based on the operating environment of the agent undertaking the aggregation. The following technical challenges are significant in the context of Service Lifecycles:

- Service Creation: Creating a service description using a standard format is an important requirement—to enable the service to be subsequently discovered. The creation of a service also necessitates associating the service with an agent. An agent would receive a request for an application service and create a new instance of it using the Factory Interface [7]. Each agent therefore provides a persistent place holder for an application service. An important challenge in this context is determining the number and types of services that should be managed by a single agent.
- Service advertising and discovery: Registering a service with the local community manager may restrict access—unless there is also some mechanism to allow community managers to interact. Discovering a service across multiple network based registries becomes an important concern—and efficiency of the referral and query propagation mechanisms between community managers become significant. The greater the number of participants that need to be contacted to search for a service, the more time consuming and complex the search process will be. The number of registries searched to find a service of interest becomes an important criteria, as does the mechanism used to formulate and constrain the query. The ability to divide a query into sub-parts which can be simultaneously sent to multiple registries is useful in this context—although it restricts the specification of a query.
- Contract enforcement: The community manager is responsible for ensuring that a request for service provision is being honoured by an agent within the community. There is a need for monitoring tools to verify that a contract is being adhered to—although this requires an agent to reveal its internal state to the monitoring service. Enforcement of a contract also requires the community manager to de-register the service or to restrict access to it if it does not meet its contract. As previously discussed, it is also possible for a community manager to change the risk or reputation index of such a service—and utilise monitoring tools to periodically update this. Contract enforcement must be undertaken based on a community specific policy.

A service may also register interest in one or more event types via its agent or the community manager. Certain event types may be common for all services within a community, and handlers for these provided at service creation time. Such an event mechanism may also provide support for service leasing—whereby a service is

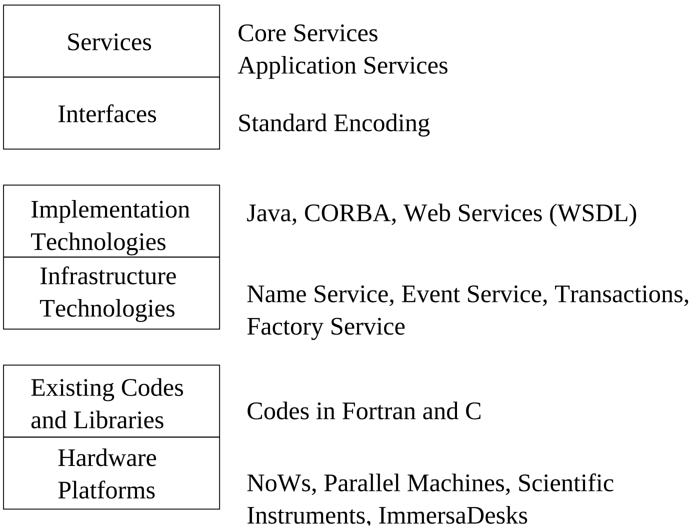

Services
Core Services
Application Services
Interfaces
Standard Encoding
Implementation Technologies
Java, CORBA, Web Services (WSDL)
Infrastructure Technologies
Name Service, Event Service, Transactions, Factory Service
Existing Codes and Libraries
Codes in Fortran and C
Hardware Platforms
NoWs, Parallel Machines, Scientific Instruments, ImmersaDesks

Fig. 4.1. *The Services Stack*

only made available to a community (or to external agents) for a lease duration—the lease is monitored by the community manager. When the lease period expires, the service agent must either renew the lease or delete the service.

4. Service Types and Instances. Figure 4.1 illustrates the layers within service oriented Grids—starting from the services themselves (which can be infrastructure or user services) and interfaces to these services encoded in some agreed upon format. At present no standard exists within the Grid community, although there are working groups in the GGF [11] exploring standard interfaces for services within a particular application domain. Existing work on the Common Component Architecture (CCA) [10] provides a useful precedence for developing common interface standards. Some of the services may also wrap existing executable codes, developed in C or Fortran—requiring the users of these legacy codes to publish interfaces to their code.

Services may subsequently be implemented using a number of different technologies—and interface definitions using WSDL may bind to a number of different implementations. Service interaction is then supported through an infrastructure that provides support for service registration and discovery, distributed event delivery between services, and support for transactions between services. Currently, this is provided by systems such as Globus, although the need for integrating such infrastructure services from other platforms, such as Enterprise JavaBeans or CORBA becomes significant.

Services are assumed to be of two categories: (1) infrastructure services provided via Globus/OGSA (for instance), and (2) application services provided by agents. Examples of infrastructure services include a Security Service, an Accounting Service, a Data Transfer service etc. Examples of application services include Matrix solvers, PDE Solvers, and complete scientific applications. Agents utilise infrastructure services on-demand, and may use type information made available by infrastructure services. Agents can also interact with each other based on a goal they are aiming to satisfy.

A minimal set of service metadata should be agreed upon by all agents within a community, regardless of the application domain—referred to as a “Services Ontology". Such an ontology would be used by agents to discover other service providers and service consumers, and the types of services they offer—and based on the Grid Services Specification (GSS) [22]. Terms within such an ontology can include the concept of file/service title, authors/service manager, locations, dates, and metadata about file content—such as quality, provenance etc. Each agent responsible for a service must also decide how to process requests being made to a given service that it manages. These criteria may be enforced by the community manager, or based on the attributes of the services being managed by the agent.

4.1. Service Interactions and Communities. Interactions between services form an essential part of Grid systems, with interactions ranging from simple requests for information (such as extracting data from the Grid Information Index Service (GIIS) in Globus), to more complex negotiation mechanisms for arranging common operations between services (such as co-scheduling operations on multiple machines). Interactions between agents are constrained by the paradigm used—such as the concept of a "virtual market"—whereby agents can trade services based on a computational economy [30]. An important aspect of such an interaction paradigm is that agents need to make decisions in an environment over which they have limited control, restricted information about other agents, and often a limited understanding of the global objectives of the environment they inhabit. The concept of "communities" becomes important to limit the complexity of decisions each agent needs to make, by limiting interactions to a restricted set of other agents. In the community context, agents must be able to first establish which communities to join, and subsequently to decide upon mechanisms for making their local state visible to others. Each community must have a manager entity, responsible for admitting other agents, and for ensuring that agents adhere to some common obligations within the community. Interaction between agents may also be mediated via such a manager—whereby the manager also acts as a protocol translator. The community manager is also responsible for advertising the properties of a community to others, and for eventually disbanding a community if it is non-persistent.

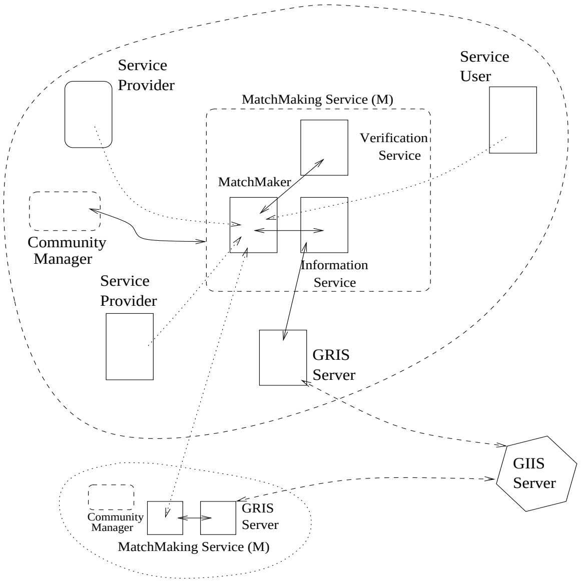


Fig. 4.2. *Service Community*

Figure 4.2 illustrates the core services provided within each community, and consists of service users/providers, a MatchMaker (M)—which is supported via a verification and information service, and a community manager. The MatchMaker provides an example of a middle agent, facilitating interaction between other service users and providers within the community. The information service can interact with the GRIS/GIIS server and locate other computational resources of interest—using the Globus system. Interaction between the service user and provider is undertaken based on a common data representation—which enables the state of a given service to be queried at a given time 't' (an example of this data model for computational services can be found in [23]). We assume that there is a single M within a community, although the request for match may utilise different criteria. The availability of a service over time extends from $t < t_{\text{current}}$ (usage history) to $t > t_{\text{current}}$ (projected usage) and includes $t = t_{\text{current}}$ (current usage). Availability over time is just one of the parameters that must be supported in the system, for instance, we also consider availability over the set of service users. The matchmaking service works as follows:

- Each Service Provider sends an asynchronous message to a pre-defined matchmaking service 'M' (running on a given host) to indicate its availability within the local community. Each message may be tagged with the service type that is being supported. The message contains no other information, and is sent to the local 'M'. The identity of M may be pre-built into each service when it is created, or may be obtained from the community manager agent (via a multicast request within the community).
- On receiving the message, the local 'M' responds by sending a document specifying the required information to be completed by the service provider agent. This information is encoded in an XML

document (see [23]), and contains specialised keywords that correspond to dynamic information that must be recorded for every service managed by the agent. The document also contains a time stamp indicating when it was issued, and an address for 'M'.

- The service provider agent completes the document—obtaining the necessary information via the GIIS server (if necessary), and sends back the form to 'M', maintaining a local copy. The document contains the original time stamp of 'M', and a new time stamp generated by the service manager. Some parts of the document are static, while others can be dynamically updated. The new service is now registered with the community manager, and can be invoked by a service user, until it de-registers with 'M'. If the service is terminated or crashes, 'M' will automatically de-register it when it tries to retrieve a new copy of the document. An alternative technique would involve a 'push' model whereby each service updates M with its state on a change. Typically, the update would be to describe changes in availability, for example after a reservation has been made by a service user. However, the update could also involve a change in capability, for example an extra service being added to the local system. If a push mechanism is used from the service to M then repeated polling of the resources is not necessary. It is useful to note that the community manager does not directly maintain any service information or content itself, and interacts with M to obtain the necessary service details.

Agents within a community may need to undertake multiple interactions to reach consensus. For instance, an agent trying to discover suitable services may need to issue multiple discovery requests before it is able to find a suitable service. Interaction mechanisms between agents therefore may be more complex, and utilise auction and negotiation mechanisms, or interaction rules. The community manager may provide mediation in this process, by restricting the maximum number of message exchanges between agents. The main objective being to enable service providers to enable their services to be more effectively used.

A particular challenge in this context is the ability to agree on a common data model for exchange service capability documents. There must be some agreement based on GSS [22], but also the ability of a service provider to identify additional properties if available in the service interface. Another important challenge is to identify the complexity of the match process (from a syntax based match to a semantic match—for instance)—and to enable a user to limit the complexity of the match in their request to 'M'.

4.2. Service Semantics. Service interactions require definitions of common terms—the definition of common units when exchanging engineering data for instance (where one service may reports its results in miles, while the service user undertakes its processing in kilometres). Service semantics are generally assumed in distributed systems—where checks on the results can be made by a user. However, when services interact directly, it is important to ensure that the results they produce follow some predefined types.

Service types may be "abstract" types—directly supported by a service, or "derived" types which are obtained by extending or combining abstract types. An agent therefore also publishes type information associated with the services it supports—enabling service users (other agents) to undertake the necessary type conversions. Service types can be based on data types supported within the service implementation—such as `float`, `string`, etc, or they may be application related—such as a `distance` type or a `co-ordinate` type. The service type mechanism may be extended into an ontology—which may also identify additional attributes, such as particular instances of types, axioms for transforming between types, and constraints on types.

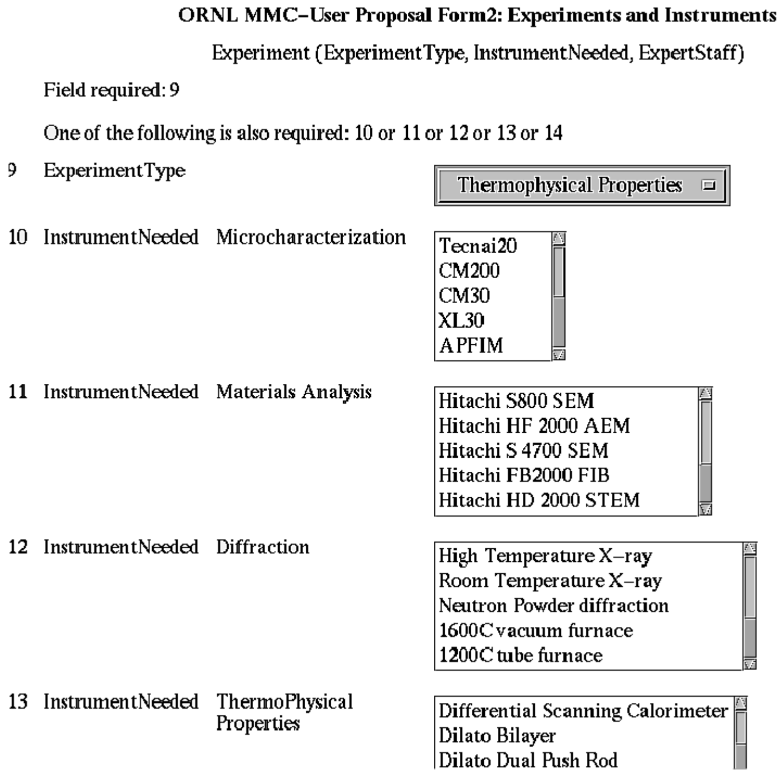
The type mechanism is also used for discovering other services, and for launching specialist services which provide a particular output type. The semantics associated with a particular type must also be defined by a service—hence, a service which uses a derived type `distance`, must prefix it with its service identity. Consequently, services with similar types but different semantics may co-exist, and can publish this information as part of their interface descriptions. One example of semantic services include mathematical libraries (such as in the MONET project [20]) with predefined categorisation of these numeric libraries. In this context therefore, a search for a numeric solver service by a user in a particular application domain would proceed by contacting one more more broker agents and perform matching based on problem domain, along with various non-mathematical issues such as the user's preferences for particular kinds or brands of software. The motivation stems from the observation that many scientists prefer to use services from particular developers, a decision often determined by the application domain of the scientist. This subjective criteria should therefore be utilised when searching for suitable numeric services—and used along with the operational interface the service offers.

In a typical Grid environment, multiple domain specific ontologies are likely to co-exist. Work being undertaken in the Gene Ontology Consortium [24] provides one example of a vocabulary being developed to support software interoperability. There are therefore likely to be a number of common services (based on a generic services ontology), and a number of specialist services (such as mathematical libraries, gene clustering software etc), which can only be invoked in a limited way, and by a restricted set of other services. An important challenge in this context is to identify the granularity at which these domain specific services should be described, and whether advertising of services should be restricted. Also important is to identify how services across domains can be defined in common ways—for instance, the use of clustering and data analysis services may be common in a number of different domains. However, the particular description schemes used may vary. Many of the concerns related to the definition of ontologies needs to be undertaken within the particular scientific community involved—although ways of identifying common services used by a number of different communities would be a useful undertaking.

5. Scenario. We illustrate the concepts outlined in this paper via a project which uses agents for managing user access to scientific instruments at Oak Ridge National Laboratory (ORNL). It was mainly aimed at automating an existing manual process of approving user requests to obtain time on a microscope and other scientific instruments. The project was undertaken as part of the Materials Microcharacterization Collaboratory (MMC) [16] project, involving ORNL and various other participants. The purpose of collaboration within the MMC is to characterise the microstructure of material samples using techniques such as electronic microscopy, and X-ray and neutron diffraction. Observation, data acquisition, and analysis are performed using instruments such as transmission and scanning electronic microscopes, and a neutron beam line. An important aspect of the MMC project is the computer co-ordination and control of remote instrumentation, data repositories, visualisation platforms, computational resources, and expertise, all of which are distributed at various sites across the US. The role of ORNL in this collaboratory was to provide access to, and management of experiments within the High Temperature Materials Laboratory [18]. A scientists is required to complete a pre-formatted proposal document (a part of this is illustrated in figure 5.1), and pass this to a central facility. Based on the type of experiment, and the instrument identified, the facility selects one or more experts to evaluate the proposal. The selection criteria involves economic factors (such as industrial impact the experiment is likely to have), technical factors (such as types of materials to be analysed in the experiment), safety factors (such as whether the user has had radiation or general training on the instrument), and credibility factors (such as what publications the user already has in the field, why the experiment is being requested etc). These factors are weighed by the expert, and a decision is made on whether the proposal to undertake the experiment should be granted. The project was conceived to automate some of the processing involved in reaching a decision on the initial proposal. It was decided that replacing the expert was not a viable option, as this would involve a detailed knowledge elicitation from existing experts, and the effort and time involved in such an undertaking would be significant. Instead, the approach adopted was to support the decision making process of the expert, and to automate as much analysis of the proposal as possible, prior to delivery of the proposal to the expert.

The automation of the current system was achieved using Web based forms, CGI scripts and an agent development tool. An agent is used to represent every entity involved in the system, and includes a "User" agent, an "Expert" agent, an "Instrument" agent, an "Experiment" agent, and two utility/middle agents, a "Scheduling" agent and a "Facilitator" agent. Each of these agents perform a pre-defined set of services, which must interact to complete the overall request. Message exchanges between agents can relate to requests for proposal to be verified, confirmation or denial of a proposal, and a verification of scheduling request. Each agent operates as an autonomous entity, in that it manages and makes requests for information to other agents, in order to achieve a given goal. The goals are specified by the physical entities which are being represented by the agent—such as a human user (for a User agent), or an instrument expert (for an Expert agent). Each agent then tries to find a set of services to be undertaken to reach the goal it has been set. Goal completion is based on each agent choosing an initial action that will lead it closer to its goal, and determined by the pre-conditions for a given action to be taken, and post-conditions (or effects) identifying the outcome of a given action on the agent itself, and its environment. The agent based approach provides the best option for modelling scenarios where a large number of users, instruments and experts can co-exist, with each entity controlling and managing its own requirements and goals.

`MatML` for Materials Property Data [25] is used for specifying intrinsic characteristics of materials. In the DeepView system developed for the MMC [27], an instrument schema has been designed for instrument properties permitting the remote, on-line operation of microscopes [28]. These schemas were examined to form the basis of a local ontology for our system. However, re-use of existing schemas raises questions concerning



ORNL MMC–User Proposal Form2: Experiments and Instruments

Experiment (ExperimentType, InstrumentNeeded, ExpertStaff)

Field required: 9

One of the following is also required: 10 or 11 or 12 or 13 or 14

9 ExperimentType: Thermophysical Properties

10 InstrumentNeeded Microcharacterization: Tecnai20
CM200
CM30
XL30
APFIM

11 InstrumentNeeded Materials Analysis: Hitachi S800 SEM
Hitachi HF 2000 AEM
Hitachi S 4700 SEM
Hitachi FB2000 FIB
Hitachi HD 2000 STEM

12 InstrumentNeeded Diffraction: High Temperature X–ray
Room Temperature X–ray
Neutron Powder diffraction
1600C vacuum furnace
1200C tube furnace

13 InstrumentNeeded ThermoPhysical Properties: Differential Scanning Calorimeter
Dilato Bilayer
Dilato Dual Push Rod

Fig. 5.1. *Form completed by the user*

the purpose and scope of an ontology within the context of an agent-based system—as our objective was to enable a user to access an instrument and performance of the system was of issue [29]. With these constraints in mind, it was decided that the concepts in the ontology must focus on use of instruments and characteristics of (human) users rather than on properties of materials such as chemical composition and geometry (MatML), and instrument characteristics such as vendor and resolution (DeepView). For these and other reasons, a domain ontology for our system was created that did not re-use concepts in the schemas mentioned above. The domain ontology is divided into four categories: Users, Experts, Experiments and Instruments—figure 5.2 illustrates the "Experiment ontology". Terms used within the ontology can take on a number of different content types—such as integers, reals, strings—and constraints are defined as ranges on these basic types. An important concern was to identify mechanisms to translate existing types supported in the form, into types that could be directly interpreted by the agents. Some attributes in the ontologies utilised by the agents required an appropriate representation of "Phase" (in the Instrument ontology), the concept of "Impact" (in the Experiment ontology), and common ways to encode time and date information. It was also necessary to constrain parameters associated with ontologies maintained by different agents—to enable interaction between agent roles.

Each agent in the system undertakes a particular set of actions to achieve its "role". A role is defined as a set of goals that need to be completed by an agent, in a given context. Hence, a User agent plays the role of an external user. In the context of the MMC, this involves "Creating a Proposal" and "Accepting a Proposal". A role is defined at a higher level of abstraction than method calls on objects, or sub-routine calls in source code. In an agent based system, a given entity (or agent) can only undertake pre-defined roles which determine its function in a given society of other agents. Hence, a User agent in this particular context cannot schedule operations on a given instrument, because it does not possess this as a role. It can make a request to a Scheduling agent to undertake such an operation, or alternatively, to communicate with an Expert agent to request a given schedule to be validated. Agents can therefore posses roles and relationships with each other based on their particular function in the agent society. It is assumed in this project that agents cannot change or modify their roles or services, although they can update the information content of their local repositories based on interactions with other agents.

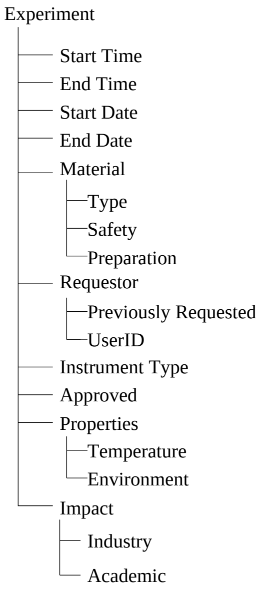


FIG. 5.2. *The "Experiment" ontology*

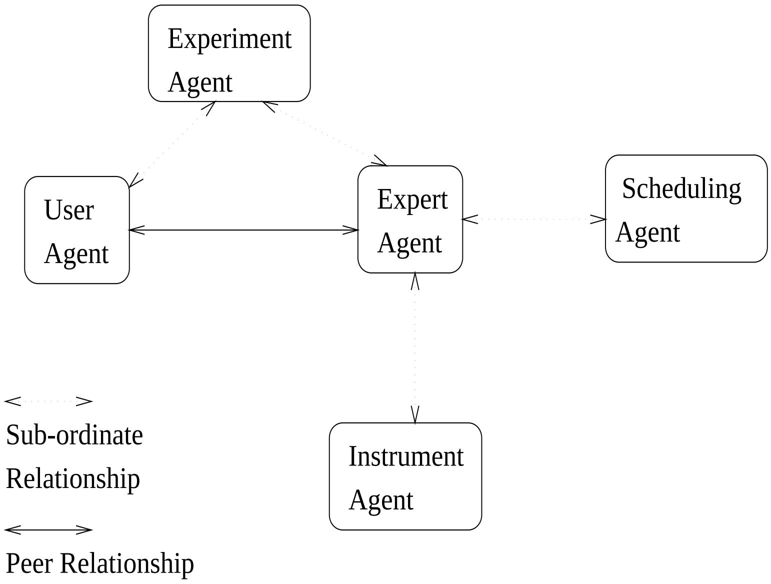


FIG. 5.3. *Co-ordination mechanism and role interaction between collaborating agents for MMC resource allocation*

A User agent and an Expert agent have a peer-to-peer relationship, as each can initiate a request to the other one. An Instrument agent is a sub-ordinate to an Expert agent, as an Expert agent can request information from an Instrument agent, but not vice versa. Roles between agents in the MMC system are illustrated in figure 5.3. Each agent in the system, and the particular services undertakes are as below:

- *User Agent:* This agent undertakes two basic services: `CreateProposal` and `AcceptProposal`. The CreateProposal task involves reading a file from disk, based on a given User ID, and initiating a proposal request to an Expert agent. The AcceptProposal task involves verifying that the schedule given by the Expert agent is acceptable—the acceptance criteria is based on checking constraints defined in the proposal with the initial request made by the User agent.
- *Expert Agent:* This agent is the most complex of all, and acts as the co-ordinator. The Expert agent can undertake one of five different services: `ReceiveProposal`, `RequestInstrument`, `CheckSchedule`, `ConfirmSchedule` and `ValidateRequest`. ReceiveProposal involves accepting a User generated request

to undertake a given experiment. RequestInstrument involves verifying constraints via the Instrument agent, based on availability of the instrument, and whether the parameters for the requested experiment are valid for the given instrument. Only two such parameters were identified as being relevant for this prototype—the "Operating Temperature" of the instrument, and the "Phase ID". Both of these are compared with the initial request from the User agent to confirm that a given instrument can support these ranges or absolute values. The CheckSchedule and ConfirmSchedule involve checking constraints on the availability of the instrument, with the availability of the expert. For the MMC, it is identified as a requirement that an instrument and an expert are available over the same time period, and that this falls within the duration of the requested experiment. The CheckSchedule task validates that such an overlap exists, and the ConfirmSchedule task generates a message to the Scheduling agent confirming the Schedule is valid. The ValidateRequest task is used by the Expert agent to confirm that a given request from a User agent does not violate any existing schedules that have already been decided. The Expert agent achieves this by interacting with the Scheduler agent, and checking the stored schedules.

- *Instrument Agent:* This agent acts as a wrapper for a microscope, and is used to identify particular access parameters required to request it for an experiment.
- *Experiment Agent:* This agent can interact with a User agent or an Expert agent to help them prepare an experiment. It supports the generation of proposals by a User agent, and the verification or checking of these by an Expert agent. Its primary purpose is to act as a support agent for helping formulate proposals, and help the User and Expert agents negotiate over parameters identified in a proposal. The Experiment agent undertakes three services: `PrepareProposal`, `CheckProposalRequest` and `ValidateProposalRequest`. The PrepareProposal task is activated by a User agent, and involves the Experiment agent helping to complete missing parameters in the proposal being sent to it. The CheckProposalRequest is used by an Expert agent to ensure that the parameters requested in a proposal are valid. The ValidateProposalRequest is used by the Experiment agent to undertake the above two services based on its local database of facts. The database is an external program that must be provided by the developer of the system.
- *Scheduling Agent:* This agent maintains a list of all valid schedules at any time, and can undertake three services: `ReceiveRequest`, `ConfirmRequest` and `ValidateSchedule`. The ReceiveRequest task involves accepting a request to verifying a proposal from an Expert agent. The Scheduling agent acts as a sub-ordinate of the Expert agent, and provides support to the Expert agent to reach a particular goal. The ValidateSchedule task involves verifying the requested schedule against its database to ensure that the requested schedule does not conflict any already assigned. The ConfirmRequest task is then used to send a message to the given Expert agent to confirm or deny the request.
- *Facilitator and Name Server Agents:* These agents acts as utility agents, mapping an agent location to its IP address (for the Name Server agent), and identifying services that a given agent can undertake, in some respects similar to a yellow page service (for the Facilitator agent).
- *Globus Gateway Agent:* The Globus/OGSA gateway agent enables an Experiment agent to launch jobs on remote instruments. Job management can be supported via the MatML data model. The gateway agent also makes use of the Facilitator and Name Server to locate and communicate with other agents.

A prototype system was implemented using the Zeus agent development tools [32].

5.1. Barriers and Discussion. Services supported by agents need to interact with infrastructure services provided through tools such as Globus/OGSA—although this is only necessary to support execution of scientific codes. Agents must therefore interact with existing Grid services via one or more gateways. Performance issues become significant when deploying agents to manage services—as no direct interaction between services exist. Existing Web services technologies—such as the use of SOAP—can have significant overheads, primarily due to the HTTP transport used and the parsing of XML based messages—especially when encoding data types along wit the content (a useful study on SOAP performance can be found in [33]). Standards such as DIME [19] may provide some performance improvement. Therefore, although the use of Web Services infrastructure may provide an important route for a wider use of Grid infrastructure, the performance implications introduced by such technologies still need to be overcome (the scientific codes currently deployed via Grid middleware have performance as a key requirement). Although many scientists may be willing to relinquish this requirement in the prototyping phase of their work—deploying production codes in this way may not be possible. Many Web Services standards are also at an early stage of development at the present time, and most experimentation is still being undertaken behind firewalls. It is also not apparent how the UDDI (service registries) are to be managed, and by whom. Should there be a few "root" UDDI registries (like current Domain Name Servers), or should the registration mechanism be more distributed? Some of these concerns need to be evaluated in the context of Grid registration services (currently utilising Globus/OGSA), to enable more effective sharing of Grid Services across applications. We also see a number of similarities between the Peer-2-Peer (P2P) approach [1] and agent systems—as both focus on service provision through a decentralised model of cycle sharing or file sharing. Whereas agent systems focus on the semantics of the shared services, the focus in P2P systems is on the efficiency of the routing mechanism used.

The use of the service oriented approach for deploying scientific codes also requires the delegation of control to a remote service. This is especially true when service aggregation is being undertaken by an agent. It is therefore important to identify how ownership is delegated in the context of such a composition process, and how a service contract must be defined and enforced for the aggregate service. One incentive for supporting such an aggregation of services may be based on the concept of a "virtual economy" [30]—whereby services can have associated costs of access and deployment. Although a useful model (and one which closely resembles the current usage of computational resources at national centres)—it is unclear how services are priced, and what roles are necessary within such an economy. Should these roles be centrally assigned and managed in the same way as index services are being used today, or can they be distributed across multiple sites? Another closely related issue is the types of relationships that must exist between services within such an economy—for instance, should we be able to support the myriad different financial trading schemes that exist in our markets, and more importantly, what enforcement mechanisms need to be provided to ensure that these trading schemes are being observed.

6. Conclusion. Issues in developing service oriented Grids are outlined. We indicate why agents provide a useful abstraction for managing services in this context, and research challenges that need to be addressed to make more effective use of agents. The need to agree upon common data models/ontologies is significant, and we view this as a significant future undertaking to make Grids more widely deployable. The need for particular application communities to agree and implement common service representations is therefore important—as is the need to agree upon a common ontology for defining generic services. A system for managing user access to scientific instruments is outlined—identifying the services supported and interactions between agents.

REFERENCES

[1] Dejan S. Milojicic, Vana Kalogeraki, Rajan Lukose, Kiran Nagaraja, Jim Pruyne, Bruno Richard, Sami Rollins, Zhichen Xu, *Peer-to-Peer Computing,* HP Labs, Technical Report HPL-2002-57, 2002.

[2] S. J. Poslad, P. Buckle, and R. Hadingham, *The FIPA-OS Agent Platform: Open Source for Open Standards,* Proceedings of Workshop on Scalability in MAS (Ed: T.Wagner and O.F.Rana), at Autonomous Agents 2000, Barcelona, Spain, 2000.

[3] David De Roure, Nick Jennings, and Nigel Shadbolt, *Semantic Grids,* `http://www.semanticgrid.org/`. Last visited: September 2002.

[4] Luc Moreau, *Agents for the Grid: a Comparison with Web Services (Part I: Transport Layer),* `http://citeseer.nj.nec.com/moreau02agents.html`.

[5] A. Avila-Rosas, L. Moreau, V. Dialani, S. Miles, and X. Liu, *Agents for the Grid: A comparison with Web Services (Part II: Service Discovery),* AgentCities Workshop at AAMAS, Bologna, 2002.

[6] M. Stevens, *Service-Oriented Architecture Introduction, Part 1,* `http://softwaredev.earthweb.com/microsoft/article/0,,10720_1010451_1,00.html`

[7] Ian Foster, Carl Kesselman, Jeffrey M. Nick, Steven Tuecke, *The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration,* `http://www.globus.org/research/papers/ogsa.pdf`, 2002.

[8] Michael N. Huhns, *Agents as Web Services,* IEEE Internet Computing, pp 93–95, July/August 2002.

[9] *The TeraGrid Project,* `http://www.teragrid.org/`. Last visited: September 2002.

[10] *The Common Component Architecture Forum,* `http://www.cca-forum.org/`. Last visited: September 2002.

[11] *The Global Grid Forum,* `http://www.gridforum.org/`. Last visited: September 2002.

[12] A. Swartz, *MusicBrainz: A Semantic Web Service,* IEEE Intelligent Systems, pp 76–77, January/February 2002.

[13] *The W3C Web Ontologies Working Group,* `http://www.w3.org/2001/sw/WebOnt/`. Last visited: September 2002.

[14] E. Turcan and R. L. Graham, *Getting the Most from Accountability in P2P,* Proceedings of First International Conference on Peer-to-Peer Computing, IEEE Computer Society Press.

[15] R. Dingledine, M. Freedman, D. Molnar, *The FreeHaven Project,* Massachussets Institute of Technology. `http://www.freehaven.net/`. Last visited: October 2002.

[16] *MMC Virtual Lab: The Materials Microcharacterization Collaboratory Project,* `http://www.ornl.gov/doe2k/mmc/`

[17] W3C, *Semantic Web,* `http://www.w3.org/2001/sw/`. Last visited: September 2002.

[18] *Oak Ridge National Laboratory, The High Temperature Materials Laboratory—User Proposal Package,* `http://www.ms.ornl.gov/htmlhome/`

[19] H. F. Nielsen, H. Sanders, R. Butek, and S. Nash, *Direct Internet Message Encapsulation (DIME),* June 2002. `http://msdn.microsoft.com/library/en-us/dnglobspec/html/draft-nielsen-dime-02.txt`

[20] *MONET:Mathematics on the Net,* `http://monet.nag.co.uk/`. Last visited: September 2002.

[21] *Web Services in WSDL,* `http://www.xmethods.net/`. Last visited: October 2002.

[22] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, *Grid Service Specification,* Open grid Service Infrastructure WG, Global Grid Forum, Draft 2, 7/17/2002. `http://www.globus.org/research/papers.html`

[23] O. F. Rana, D. Bunford-Jones, D.W. Walker, M. Addis, M. Surridge, and K. Hawick, *Resource Discovery for Dynamic Clusters in Computational Grids,* In *Proceedings of Heterogeneous Computing Workshop*, at IPPS/SPDP, San Francisco, California, April 2001, IEEE Computer Society Press.

[24] *The Gene Ontology Consortium,* `http://www.geneontology.org/`. Last visited: October 2002.

[25] *MatML Home,* `http://www.ceramics.nist.gov/matml/matml.htm`. Last visited: October 2002.

[26] Argonne National Laboratory, *The Globus Project,* See Web site at: `http://www.globus.org/`. Last visited: October 2002.

[27] B. Parvin, J. Taylor, G. Cong, M. O'Keefe, M. Barcellos-Hof, *DeepView: A Channel for Distributed Microscopy and Informatics,* Proceedings of the ACM/IEEE SC99 Conference, Portland, OR, November 1999.

[28] `http://vision.lbl.gov/Projects/DeepView/Instruments/Instrument.dtd`. Last visited: October 2002.

[29] L. Pouchard, D. Walker, *A Community of Agents for User Support in a Problem-Solving Environment* in Tom Wagner, Omer Rana (Eds.), *Infrastructure for Agents, Multi-Agents Systems, and Scalable Multi-Agent Systems,* Lecture Notes in Computer Science 1887, Springer Verlag, 2001.

[30] R. Buyya, *Economic Paradigm for Resource Management and Scheduling for Service-Oriented Grid Computing,* `http://www.buyya.com/ecogrid/`. Last visited: October 2002.

[31] Various papers—5th International workshop on *Deception, Fraud and Trust in Agent Societies,* at AAMAS 2002, Bologna, Italy. `http://wwwistc.ip.rm.cnr.it/news/wstrust.htm`

[32] *The Zeus Project,* See Web site at: `http://193.113.209.147/projects/agents.htm`, 2000.

[33] D. Davis and M. Parashar, *Latency Performance of SOAP Implementations,* Proceedings of the 2nd International Workshop on Global and Peer-to-Peer on Large Scale Distributed Systems, IEEE International Symposium on Cluster Computing and the Grid, Berlin, Germany, May 2002.

Edited by: Dan Grigoras, John P. Morrison, Marcin Paprzycki
Received: August 12, 2002
Accepted: December 13, 2002