AISB Symposium on

AI and Grid Computing

Recent advances in computer science facilitate a paradigm shift towards large scale computing – the Grid, with 'parallel and distributed' computing playing an important role. The Grid paradigm offers an important abstraction for combining national and continent wide computational resources, to enable application scientists and domain experts to work more effectively, and, subsequently, to do better science. Driven by the unprecedented increase in Internet usage, Grid-oriented computing will provide the next phase in the way we manage and disseminate knowledge. The Grid can be seen as a mechanism for integrating resources, which can range in complexity from dedicated parallel machines (maintained by national centres) to low-end clusters of workstations, connected by Beowulf/Linux, for instance. Managing such diverse computational platforms has been an active research area, and systems such as GLOBUS (Argonne) and Legion (Virginia) are often quoted as core infrastructure for managing Grid services. An alternative perspective, which subsumes the resource management aspect, involves viewing Grids as an integration of information services, which range from application specific data repositories, to data mining tools that can support the extraction of useful knowledge from such data.

Activity in Grid computing has also accelerated recently as a result of advances in information technology and its use, such as:

- Component based software development
- High speed networks
- Standardisation of interfaces to databases and data repositories
- Virtual machines and cluster computing
- Public domain and community software licensing arrangements
- On-demand (on-use) software payment schemes
- Network aware interfaces and visualisation

To make effective utilisation of resources across a Grid that spans organisational boundaries, it is imperative that the underlying infrastructure support intelligence. Intelligent software is required to undertake resource and service management, service discovery, service aggregation/decomposition, and support performance management. Commercial systems will also require the underlying infrastructure to respect site autonomy, and particular site specific policies on usage.

The objective of this symposium is to bring together researchers in computer science and AI, to discuss issues in managing Grid services and resources. Six papers are presented in this symposium, covering aspects of developing distribution mechanisms for computing, such as the paper by P. Mathieu, J.C. Routier, and Y. Secq on "RAGE". Another aspect discussed is middleware to connect Grid based applications, such as the paper by S. Newhouse et al. on ICENI,

work by Brazier et al., and by M. Allen et al. on Jini based systems. A third aspect is the availability of software development libraries, such as "JFIPA" by Amund Tveit, and the need to deploy agents on a large scale (especially in the context of the emerging area of Web Services), as described by S. Thompson for the European "AgentCities" project.

We are grateful to the review committee for participating, and included:

- Luc Moreau, University of Southampton
- Wolfgang Emmerich, UCL
- David Walker, Cardiff University
- Rajkumar Buyya, Monash University, Australia
- Robert Allan, Daresbury Laboratory
- Ian Wakeman, University of Sussex
- Jim Austin, University of York
- Steven Newhouse, Imperial College, London
- Julie McCann, Imperial College, London
- Philippe DeWilde, Imperial College, London
- Steffen Moeller, University of Rostock, Germany
- Cefn Hoile, BT Labs

Michael Schroeder (City University, UK)
and Omer Rana (Cardiff University, UK)
March 2002

Adaptation Engine: an Agent-Based Framework for ad-hoc Service Life-Cycle Management for Meta-Computing

M. Allen, E. Grishikashvili, N. Badr and A. Taleb-Bendiab
School of Computing and Mathematical Sciences
Liverpool John Moores University
Byrom Street, Liverpool, L3 3AF, UK
{m.allen, cmsnbadr, cmsegris, a.talebbendiab}@livjm.ac.uk

Abstract

The cluster and grid-computing paradigm offers support for large scale, widely distributed, high-performance computational systems. Several of such architectures and frameworks have been developed aimed at primarily large parallel computations in support of scientific, engineering calculations and problem solving. With the emergence of the "service-oriented" business pattern for low-cost e-business, and increasing users' need for high-volumes of multi-media contents and applications, we argue that the rapidly maturing grid technology will offer commercial opportunities for the development of a range of systems, infrastructures and services to support high-utilisation and availability of global computing and data resources for widely distributed enterprises. However, before this becomes a reality much research work is required to address a number of well documented grid issues including; ubiquity, high-assurance, flexibility, ease of use and alignment with emerging industrial standards. In this paper we argue for the potential of new distributed systems, aimed at e-business and commercial enterprises, leveraging the strands of research from cluster computing, grid computing, agent systems and data description languages. Based on an on-going research work, which focuses on high-assurance composable systems engineering, this paper will outline a service-oriented approach and associated agent description languages, which are used to facilitate the construction and management of ad-hoc federated software services.

1. Introduction

Over recent years the meta-computing concept and technology has been "rediscovered" by wider-based academic and business communities. Their interests, whilst encompassing the need to access high-throughput computing facilities, are driven by commercial pressures: to reduce the cost of ownership of data-intensive applications and high-volume data storage systems, and to improve utilisation of organisational assets including; computational, data and physical resources.

Many grid and e-science initiatives are now underway, based on meta-computing systems, such as Globus (Foster I. 1997), Legion (Natrajan A. 2001), Information Power Grid (Grid), NetSolve (Casanova H.), Ninf (Ninf), AppLes (Project), Nimrod (Buyya 2000), DISCWorld (Hawick K. et al 1999), Unicore (UNICORE), EcoGrid (EcoGRID), and iGrid (Brown) to provide frameworks, middleware architecture and user applications to ease and facilitate communities access to widely distributed high-performance computational resources, high-cost scientific facilities and data resources.

A detailed review of these projects and systems is out of the scope of this paper, the reader is referred to the above references. However, in this paper we list the five key grid categories[1] which are detailed in Foster and Kesselman (Foster I 1999);

- Distributed Supercomputing: Where a grid of clustered computers is formed enabling the utilisation of under utilised computers, resulting in cost-effective access to a super computer capability. For example the Beowulf system (Ridge D. 1997), PVM (Sunderam 1990) and MPI (MPI) were developed to support clusters.
- High-throughput computing: Where loosely coupled tasks can be farmed to a widely distributed system exploiting unused processor cycles[2]. Some examples are the SETI@home (SETI@home) and the THINK (Project) projects
- On-demand computing: Which facilitate the provision of scarce distributed resource on a just-in-time basis such as; providing software applications network services, memory and/or storage space when needed.

[1] Some of the resource management problems are discussed in [12].
[2] These clusters or grids are used to explore and exploit distributed super computing and high-throughput computing.

- Data-intensive computing: Which facilitate access and aggregation of new information from geographically distributed data sources.
- Collaborative computing: Which enable distributed workers or virtual team workers to co-operate and share computation resources.
- Storage Computing: A Storage Area Network (SAN) (Network) is a high-speed special-purpose network that connects different types of data storage devices with associated data servers on behalf of a larger network of users. Typically, a storage area network is part of the overall network of computing resources for an enterprise. A storage area network is usually clustered in close proximity to other computing resources but may also extend to remote locations for backup and archiving, using wide area network technologies.

2. Middleware

However, in view of the distributed heterogeneous nature of grid computing nodes including their software and network systems management, and user management mechanisms, a middleware approach is currently adopted in a number of projects underway to provide "seamless" access, integration and co-ordination of dispersed grid resources and executions (processing jobs).

There are several middleware standards and supporting technologies emerging including; CORBA (CORBA), EJB (Microsystems), Jini (Microsystems), Web Services (E. Christensen 2001), JXTA (JXTA), DAML (DAML). These standards can be leveraged through middleware to enable grid solution development, and deployment and their seamless integration with enterprise systems including legacy systems. In addition the capacity of XML to provide machine-understandable data description completely independent of language, platform and environment greatly enhances the possibilities for large scale, widely distributed and heterogeneous systems.

3. Agents and the Grid

Grid computing offers the infrastructure to support distributed computing. The predicated behaviours of software agents, such as agility, flexibility, responsiveness, adaptation, autonomy and learning, offer even greater benefits for low-cost and low-maintenance computing. Integrating the two paradigms would therefore seem to be a proposition with potential for great benefits to commercial and non-commercial domains. In response to these assertions a number of research works are currently proceeding to investigate the application of software agents within, or supporting, a grid environment.

We model our agents as service providers and consumers. Each communicating with other agents to acquire access to resources or services they need to achieve their own goals. These services vary from managing access to scarce or expensive resources, both computational and physical (such as mathematical equations or theorem provers), to providing information as to the current state of the grid, to managing the life cycle of applications.

Software agents can take on a number of different roles within a grid environment including:

- *Information agents:* Where software agents gather and supply, to users or other agents, information on the current state of the grid.
- *Wrapper agents/adapters:* For integrating legacy system and data.
- *Proxy agents:* For representing users in the system.
- *Management agents:* Which manage computational execution (job), and coordinate and manage other agents and systems.
- *Resource controllers:* To control access to scarce resources. Within a grid environment an autonomous agent monitors the execution of a job, through peer-to-peer communication with other agents that collaborate to execute and complete the job. In essence this means that some of the management tasks of a grid architecture can be performed and/or encapsulated into collaborative agents operating in the grid environment; leading to a decentralised peer-to-peer model of administrative organisation and management.
- *Computational agents:* That carry out the users computations.

4. Agent Grid Frameworks

Cougaar (DARPA 2001) is an architecture developed by DARPA, to provide large-scale cognitive agents architecture. The Cougaar project recognised the inadequacy of existing software-engineering techniques for building highly complex systems involving many thousands of different objects. Originally developed to support military logistics it has potential to be used in a wide range of modelling and development domains.

CoAbs (CoABS) is another DARPA funded project concerned with building agent oriented grid computing. The project is investigating various aspects of agent computing including robustness, team computing and service description languages. Both the CoAbs and Cougaar projects are aligned with DARPA Agent Mark-up Language (DAML) initiative (R. Medvidovic 1998). The latter is leveraging XML and RDF to create a range of machine understandable agent and agent services description languages and ontologies advocated for the semantic web vision (web). For a full description of the initiative the reader is referred to the DAML website (DAML).

Williams and Talib-Bendiab (MJ. Williams 1998), described an architecture independent multi-agent framework, which was developed in Python programming language for CORBA compliant adaptive software agents. Where the developed environment supports dynamic configuration, where both agents and users can re-configure the system to as fine a granularity as is needed (or is authorised). The framework provided a set of modules including:

- ***Generic Agent Shell***: This module provides the base operations an agent (in this system) can provide. They include: an extended contract-net negotiation protocol, a simple communications language, a service executor/scheduler, a configuration language, an internal monitor process, a security module, and a reasoner. The shell also allows any module to be replaced so that other reasoning paradigms/scheduling methods can be used at a later date.
- ***Agent Definition Language***: Agents are viewed as intelligent service providers. Using ADLe we define agents in terms of the services they provide. These services are then defined in terms of the atomically executable tasks that make them up. These tasks are described in an interpreted language.
- ***Agent Re-Configuration Language:*** This script based language instantiates agents defined in ADLe earlier with a behaviour and binds them to their respective machines/resources. This configuration is performed at runtime, meaning that agents can reconfigure themselves on demand. The configuration language also provides a means to update tasks, services and agent configurations at runtime (both programmatically and interactively). A generic shell is wrapped around the instantiation providing the agent with negotiation and communication skills.
- ***Communication wrappers***: The agent shell (and therefore the agent itself) is unaware of the communication method it is using. Currently wrappers have been developed that allow the agent to communicate using sockets, CORBA (CORBA) or via WWW Servers. It is up to the developer to choose which one their agent uses. The CORBA based wrapper is implemented on top of Prism Technology's OpenBase and OpenOrb (Technology), a CORBA compliant distributed object platform.

In the remainder of this paper, we outline an application of a software agent framework (MJ. Williams 1998) to support rapid creation, monitoring and modification of middleware support services for ad-hoc marketplace collaboration..

In the next section we outline a set of description languages. The supporting application is currently being ported from Python to the emerging web services standard.

The ADLe, a snippet of which is illustrated in Figure 1, describes an agent in terms of the services it provides, the services it requires (from other agents) and the resources it requires. This idea permits the invocation of an agent at run time that takes on the characteristics it needs to complete its allotted task.

The agent is composed from the XML descriptions created by the user. This environment provides for the development of an agent-based invocations service. This will allow seamless access and management of federated meta-computing resources and services (R. Medvidovic 1998). Software agents are used to discover, supply and manage the lifecycle of any specific service cluster (federation) that is required to provision a given marketplace contract.

5. Agent Definition Language

As shown in Figure 1, outlining a "snippet" XML-based Agent Description Language (ADLe), an agent is modelled as both a service provider and consumer with access to local resources, such as code and environment required to perform its execution. For instance, an agent of name ***name,*** provides a set of services ***provide_services_set***, and it uses a list of other services ***use_services_set*** and requires a specified ***resources_set.*** Also, an agent's service binding can be negotiable or not to enable a Jini-like leasing and renewal of contracts as required by the contract net protocol (Smith 1980.).

```
<?xml version="1.0" encoding="UTF-8"
?>
<!DOCTYPE Agent Inventory System
"agent.dtd">
```

```
<agent-definition>
<comments>This is a
comment</comments>
<version>1</version>
<class_def>extend_class</class_def>
<agent-defn-name name="agent" />
<provides>service-
description</provides>
<uses>service-description</uses>
<requires>
<resource-
description>resources_set</resource-
description>
</requires>
</agent-definition>
```

Figure 1: An example of an agent meta description

In addition, in this framework, we define resources (Fig. 1) as core and/or low-level computational resources required by an agent to operate in its grid environment. For example the services own codebase (software components and references to jar, dll, etc) and local computational resources and environment attributes. The resources description is represented using the emerging W3C Resources Description Framework (RDF) (Framework)

6. Service Description Language

The services are defined in terms of the tasks or services required to perform them in order to satisfy the request. A task is atomically executable from the point of view of the agent executing it. If a service uses a task then it will be local to that agent; if it uses a service then it will tend to be a service provided by another service provider including agents and web services.

```
<?xml version="1.0" encoding="UTF-8"
?>
<!DOCTYPE Service Inventory System
"service.dtd">
<service>
<comments>This is an example of a
service description.</comments>
<version>1.1</version>
<service-description
name="provide_service_set"
negotiable="yes" type="public"
AAA="certificate">
<execution-model model="sequence"/>
<collection>
<task-list task="task_1"
negotiable="yes" type="public"
AAA="certificate"/>
<task-list task="task_2"/>
<task-list task="task_n"/>
</collection>
</service-description>
</service>
```

Figure 2: An example of a service meta description.

Services are listed in the form of their XML tags. Attributes, following the 'service' tag, define the behaviour of the service. ***nonnegotiable*** means that an agent will provide this service at all times (i.e. it is not negotiable). The ***public*** and ***private*** attributes tell the service executor whether the service is a local service for internal use. The service definition defaults to being *negotiable, public* and *non-concurrent*.

The service definition is interpreted by the service executor when a request for that service comes in (if it is agreed upon). It is executed as a transaction that fails if any of the tasks within it fail. Within the service there are multiple levels of transactions allowing the ***sequence***, ***parallel*** and ***try*** statements to also be transactional. Nested ***parallel***, sequence and ***try*** commands allow a degree of control over how the tasks will be executed. ***parallel*** executes a set of tasks in parallel and ***sequence*** in sequence. The ***try*** statement tries each task in turn until one succeeds at which point it exits successfully. If all the tasks within the statement fail then the whole ***try*** fails and so the transaction/service itself fails.

Tasks were originally defined in Python, an object-oriented, interpreted language. An interface between Python and the C++ shell was provided by the use of the Python API, embedding the Python executor in the shell. As part of the updating we are now porting the program and tools to Java but the use of meta languages to describe agent behaviours aids language independence.

7. Service Assembly Process

The agent that begins and manages user-entered jobs acquires its list of activities from the users' script. This script defines the tasks to be performed, the location of user data, and the output destination. Output may be a file, or files, of transformed data or a visual presentation tool. The script is encoded into XML, or some XML defined schema, and is produced by a visual tool. This transforms the users requirements into the XML script, developed from the Agent (Re-)Configuration language defined in (MJ. Williams 1998).

After the definition level, we have a set of agent 'classes' available to instantiate. Agents that provide the same services should have the same agent definition. It is only upon instantiation, when they are given a behaviour that they will differ.

There are two ways the user can configure the system, at compile time or at run time. At compile time the initial system configuration is placed in a script that starts the system up. At run time the user connects to a local configuration manager and types configuration

commands interactively. At run time, agents are also given the ability to configure themselves. This allows them to be much more adaptable to their environment.

The commands available to the user are:

- *inst* - This creates an instantiation of an agent class at a particular machine (that has to be running a configuration manager as well). It also tells the configuration manager from which file it should pick up the behavioural description from.
- *form group* - This groups a set of agents. This forms a secure group that only accepts reconfiguration commands from within itself.
- *form composite* - This also groups a set of agents. However, it differs from the *form group* command in that externally the group appears to be a single agent.
- *update* - This allows a service/task/behaviour on an agent to be updated 'on-the-fly'. The update occurs immediately, but any outstanding contracts using the previous service/task will be honoured using the old version.
- *merge* - This command merges two agents. Currently this is implemented using a version of the *form composite* command.
- *migrate* - The specified agent is moved to a new machine.
- *pass* - This allows knowledge or service/task definitions to be passed between agents. An argument specifies whether the information is to be removed or not from the source. When the user is controlling the configuration process it allows the addition of new services to agents.
- *authorise* - This command provides access to the authorisation database within an agent.
- *set* - This command gives access to various debugging controls on agents in the system (not described here).
- registration – This command registers the agent with a service advertising service.

The two languages described above -- Agent Definition Language and Agent re-configuration Language, will underpin the development of our agent-based adaptation middleware services, which will be applied in the context of re-configurable meta-computing.

8. Agents in a grid architecture

In order to operate effectively agents require information. Where are the resources that the agent needs to carry out its instructions? An agent that supplies the specified resource as a service manages the resources themselves. Finding the service agent clearly requires a centralised information service such as the Grid Information Service that is part of the Globus project (Foster I. 1997). Or at least access to the information service must be through well-known published addresses – the information itself can be distributed throughout the grid.

In the grid architectural diagram from (Foster I. 2001) five layers are shown. These are:

1. Problem Solving Environment (The human interface).
2. Middleware Common Services
   - Applications and supporting tools.
   - Application development Support
3. Grid Common Services
4. Local resources (The fabric of the grid).

Our agents inhabit layers 1 and 2. They aid the development of application software by providing runtime choices for the elements of the application development support layer. This layer includes MPI, CORBA, Condor, Jini and OLE/DCOM services but our agents are not constrained to a particular communications solution. The agent contacts the layer 2 service managers to arrange connections to level 3 services. The protocol or mechanism chosen for this is transparent to the initiating agent and consequently to the application developer. This is a low-level resource described in the ADLe and Resource Definition.

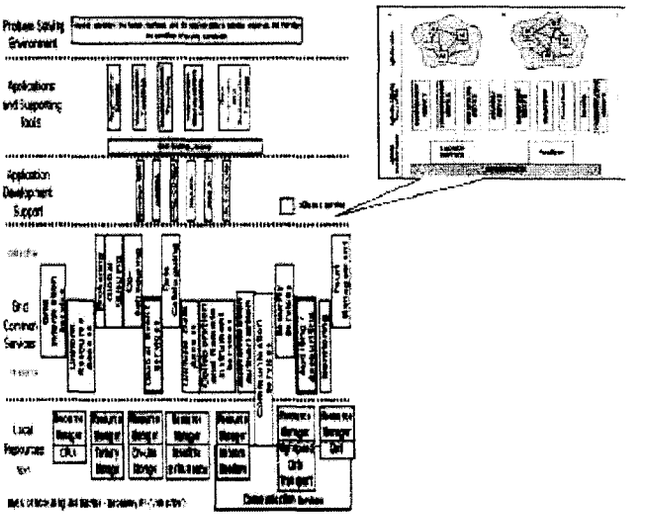


Figure 3: Grid Architecture

An important ingredient of a long running and robust grid is a capacity for on-the-fly re-configurability. This applies mainly to changes and additions that occur over time and not by the change in the grid topology caused by node loadings or network failures. This is already handled by the local management substrate built into the agents. A toolset to support dynamically re-configurable multi-agent system is described in (MJ. Williams 1998).

9. A Configuration Scenario

For the purpose of this paper, a scenario is described below, which is based on an industrial case study conducted with a UK aerospace organisation. The study focused on business process modelling to support e-work and resource sharing across the company's global network of design services. In particular, aerospace engineering design work is chosen for this scenario[3]. The workflow for a user to define and submit a given job to the job manager includes:

1. Define and submit a given user's job including its associated constraints, input data sources, output destinations, job logging and error handling information.
2. Interpret the job description (script) and instantiate a job manager agent to organise and carry out the required tasks.
3. A job management agent needs to be kept informed of the jobs' progress or of any problems that occur while the job is running. For example if a node running a requested service goes down. This requires monitoring activity on behalf of the job management agent. Two elements will know the current status of a running service agent, the agent itself and the agent platform under which it runs. Only the agent itself can know the status of the job(s) it is running therefore in the first instance monitoring will be in the form of peer to peer communication between the job management agent and the service provider. However, an agent that is in difficulties, because of some computation that has caused an internal malfunction, may not know of its problems or maybe unable to reply to a job progress enquiry. In this instance communication with the agent platform would be required.
4. Delegate: an agent, representing a user/job pairing, reads the generated script and sets out to discover if the resources needed are available and where they can be found. This agent communicates directly with the agents that supply the required services and requests that the service be provided – and supply relevant information.
5. Serve and compute: The service agents carry out their tasks directing results either back to the job management agent or to a pre-determined storage area. When a service agent completes its allocated task it informs the job management agent.
6. Inform: When all tasks are complete the job management agent informs the user in the case of non-real time computation – or directs the results to the users interface program.
7. Generate report – directs the results to the users interface program and/or generate a report document, which can be accessible through a portal technology.

The specific task focused on agent based support for distributed design teams.

- To distribute the designers' work to other members of the design team and or job farm. For instance, it provides is distribution of designers' daily work to other team members. This service is made up of two sub services, which are to gather the local data, the latest versions of documents, and transmit the information that these documents have altered to the other proxy agents. The service that the agent requires is the address look up for the agents it contacts, the communication sub service.
- To gather information design and management information from a legacy data repository and other sources.
- To act as a sentinel, looking over the designers shoulder to seek out similar documents that the designer is working on and present these documents to the designer with a persuasive argument as to why he should view them.
- To perform engineering and costing calculations.
- To transparently manage the designers' use of physical and computational resources.

In a peer-to-peer mode, these proxy agents representing the team member and other serve/compute agents are formed into a group. Thus enabling group-based communication and control of resources and re-configuration. At a more detailed level the steps of the agent's life cycle are:

- Load ADLe (uses GAS and Data Repository).
- Load defined behaviours (uses Behaviour Service).

[3] Since it relies on high computational resources, high-value commercial and proprietary engineering design software, access to high volumes of data and knowledge sources, and data storage and data visualisation facilities.

- Load authorisation (from authorisation service). The execution capability of any re-configuration and/or change is controlled by group owner authorisation and/or its delegation to other authorised agents. The user or system manager initiates the change, either by sending an *update* command, which contains a file address for the new behaviour. This may be either a change to a current behaviour or the addition of a completely new behaviour.
- Registers service with advertising service.
- Forms a composite (this is the group of agents representing the designer).
- Run code loaded from behaviours (each behaviour in its own thread).
  - Get the list of folder/directories to check for document changes.
  - Get the time to check for document changes.
- Check Documents.
- Get proxy addresses.
- Transmit Changes.

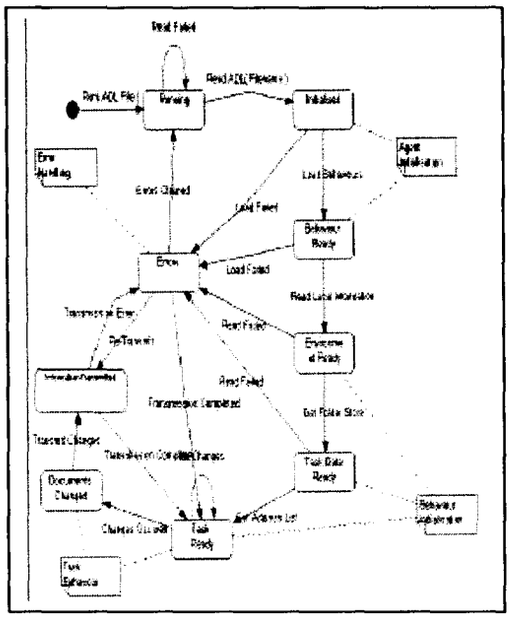


Figure 4: State chart of Designer Proxy Agent

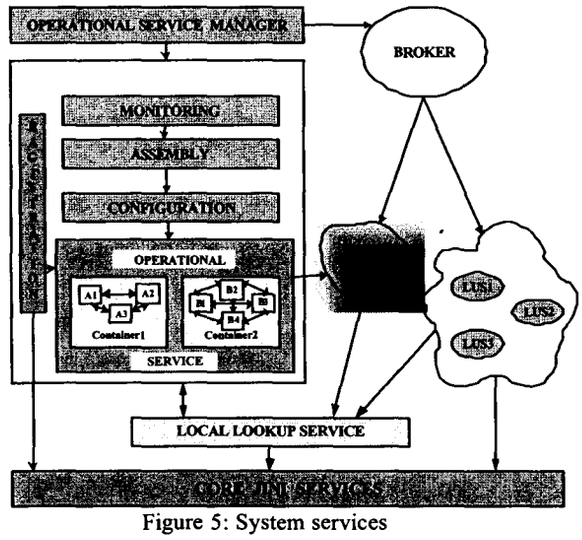


Figure 5: System services

- The architecture (as shown in figure 5) provides services for monitoring the system, assembling and configuring agents, registering services, grouping agents and providing primitive services, initially through Jini core services. Resource and Service Repository: Above we discussed the need for information. Agents require information to make decisions about how and where they undertake their actions, as well as which services are available and where they can be acquired. In (Enrich01 2001) we describe the development of an Inventory Management program to administer legacy data used to support virtual teams of new product developers. This system will be developed further to become an information provider (the Grid information service of level 4) for the agents operating in the grid environment.

- Dynamic Class Loader (Behaviour Loader): Existing agent behaviours, as encoded into the ADLes will also be stored and managed through the repository. The repository makes use of XML encoded meta data to provide services to those requiring the stored information. This can be agents requesting behaviours as well as humans.

- Scripter: A graphic tool to allow users to describe agent behaviours, services and resources.

- Interpreter/parser The information here, encoded into DAML+OIL, is information advertising the services available to the grid and the status of those services at the current time.

- Dynamic Topology Manager (maybe including the configuration Manager)
- Agent Resources manager (including broker and matchmaking) The expanded Inventory Management (IM) system, which is now a Resource Information Broker and Repository (RIBR), is itself a grid of service providing agents. It is currently implemented through Java servlets acting as a portal to the repository with custom SOAP (SOAP) messaging.

10. Conclusions and Future Work

The meta-computing paradigm offers support for large scale, widely distributed, high-performance computational systems. Several of such architectures and frameworks have been developed aimed at primarily large parallel computations in support of scientific, engineering calculations and problem solving. With the emergence of the "service-oriented" business pattern for low-cost e-business, and the increasing users' needs for high-volumes of multi-media contents and applications, we argue that the rapidly maturing grid technology will offer commercial opportunities for the development of a range of systems, infrastructures and services to support high-utilisation and availability of global computing and data resources for widely distributed enterprises. However, before this becomes a reality much research work is required to address a number of well documented grid issues including; ubiquity, high-assurance, flexibility, ease of use and alignment with emerging industrial standards. In this paper we argue for the potential of new distributed systems, aimed at e-business and commercial enterprises, leveraging the strands of research from cluster computing, grid computing, agent systems and data description languages. Based on an on-going research work, which focuses on high-assurance composable systems engineering, this paper will outline a service-oriented approach and associated agent description languages, which are used to facilitate the construction and management of ad-hoc federated software services.

The updated system, translated into Java and making use of more recently developed meta languages, will provide a cross platform, experimental architecture for exploring the development of robust, agile and adaptive middleware in support of virtual teams and commercial, e-business activities whilst allowing use of existing protocols and applications.

The growing maturity and understanding of grid and agent based systems offers potential for the above in the search for reduced cost computational support.

Our agent description languages were originally developed before the emergence and acceptance of XML, RDF, OIL and DAML. Thus, we are currently reviewing DAML and the DAML "dialect" to develop a description language translation service from our developed languages to DAML and WSDL.

Acknowledgements

This work is financed by the UK EPSRC (GR/M02958) under the Systems Engineering for Business Process Change (SEBPC) initiative and EPSRC CASE award in collaboration with Prism Technologies Ltd, Gateshead. Thanks are also due to the project academic partners at the Centre for CSCW, Lancaster University.

References

AppLes Project – http://apples.ucsd.edu/

Brown, M et al, The International Grid (iGrid): Empowering Global Research Community Networking Using High Performance International Internet Services, http://www-fp.globus.org/documentation/papers.html.

Buyya, R., Abramson, D, and Giddy, J., Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid, HPC ASIA'2000, China, IISEE CS Press, USA, 2000.

Casanova H., Dpongarra J., (1997) *Netsolve: A Network Server for Solving Computational Science Problems,* International Journal of Supercomputing Applications and High Performance Computing, Vol. 11, No. 3

CoABS http://coabs.globalinfotek.com

CORBA http://www.corba.org/

DAML http://www.daml.org/

DARPA (2001). Cougaar, BBN Technologies http://www.bbn.com/abs/caa.html

Erik Christensen, Microsoft Francisco Curbera, IBM Research Greg Meredith, Microsoft Sanjiva Weerawarana, Web Services Description Language (WSDL) 1.1*W3C Note 15 March 2001* IBM Research

EcoGRID - http://www.csse.monash.edu.aul-rajkumarlecogrid/

Enrich project http://www.cms.livjm.ac.uk/erich/

Enterprise Java Beans http://java.sun.com/products/ejb/

Foster I. and C. Kesselman (eds.). The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann, 1999.

Foster I., Kesselman C., (1997) *Globus: A Metacomputing Infrastructure Toolkit,* International Journal of Supercomputer Applications, 11(2): 115-12

Foster I., Kessleman C., Tuecke S., *The Anatomy of the Grid,* International Journal of Suoercomputer Applications, 2001

Grid Protocol Architecture Working Group Johnston W., http://www-itg.ibl.gov/GPA

Hawick K. et al, DISCWorld. An Environment for Service-Based Metacomputing, Future Generation Computing Systems (FGCS), Vol. 15, 1999.
Information Power Grid http://www.ipg.nasa.gov/
JXTA http://www.jxta.org/
JINI http://www.sun.com/jini/
MJ Williams and A. Taleb-Bendiab, 1998, *A Toolset for Architecture Independent, Reconfigurable Multi-Agent Systems*, Proceedings of First International Workshop on Mobile Agents, Lecture Notes in Computer Science, Springer-Verlag, 1998, pp210-222.
Message Passing Interfac
http://www-unix.mcs.anl.gov/mpi/
Natrajan A., Humphrey M., Grimshaw A., *Grids: Harnessing Geographically-Separated Resources in a Multi-Organisational Context* High Performance Computing Systems, June 2001.
Ninf – http://ninf.etl.go.jp/
OMG CORBA http://www.omg.com/
Prism Technology http://www.prismtechnologies.com/
Richard Medvidovic N. Taylor David S. Rosenblum Software Architecture and Component Technologies: Bridging the Gap Peyman Oreizy Nenad Workshop on Compositional Software Architectures Monterey, California January 6-8, 1998
Ridge D., Becker D., Merkey P., Sterling T., *Beowulf: Harnessing the Power of Parallelism in a Pile-of-PCs* Proceedings, IEEE Aerospace, 1997
Resource Definition Framework http://www.w3.org/RDF/
R. Smith. *The contract net protocol: High-level communication and control in distributed problem solver*. IEEE Transactions on Computers, 29(12):1104--1113, December 1980.
Semantic web http://www.semanticweb.org/
SETI@home http://setiathome.ssl.berkeley.edu/
Storage Area Network INM Redbook sg24547
SOAP http://www.w3.org/TR/SOAP/
Sunderam V. S., *PVM: A Framework for Parallel Distributed Computing,* Concurrency: Practice and Experience, 2, 4, pp 315--339, December 1990.
THINK Project http://members.ud.com/home.htm/
UNICORE - http://www.unicore.org/

The Agentcities Network & Other Information Spaces

Simon Thompson
The Agent Team
Intelligent Systems Lab
BTexact Research
Ipswich.

An Agentcity[1] is an agent platform that conforms to a subset of the FIPA interoperability open standard and has been federated with the Agentcities network to form a part of a global information space. Each Agentcity is intended to be open, providing a forum for the registration and location of agents from diverse service providers. In this paper we describe the Agentcities network, and discuss how Agentcities is related to the Semantic Web, Web Services and The Grid and point to the challenges that face the Agentcities effort in the future.

1 Introduction

The Agentcities initiative has developed out of the FIPA agent standards body with the objective of deploying a world wide network of agent platforms. In Willmott, Dale, Burg, Charlton, & O'Brien 2001 the group responsible for the initial Agentcities vision outline their intentions and plans for the development of the Agentcities network. In this paper the direction that the initiative is taking in the personal view of the author is described and some of the challenges that are being generated by the attempt to build a world wide agent system are outlined.

1.1 The Agentcity Philosophy

The philosophy behind Agentcities can best be summed up with the phrase "structured diversity is superior to imposed unanimity" (Greaves 2001). Open networks are not useful if we insist that they are used for one purpose in one particular way. Instead we want systems like the Web, which was developed for the purpose of exchanging information about high energy physics, and has evolved into a multi-use, e-learning, intranet, news distribution and e-commerce platform.

1.2 The Agentcities Initiative

Agentcities is "an initiative to create a global network of agent platforms and services to which researchers can connect their agents" (Willmott et-al 2001). Agentcities is based on the work that has been done by the FIPA agent community recently. These activities have produced two key resources that have enabled the initiative to be kick-started:

- A set of consensual, relatively mature normative standards developed by a community in a concerted long term effort
- A body of open source software, including high level reasoning tools like JESS, agent toolkits like Zeus and PARADE, agent platforms like AAP, JADE and FIPA-OS and parsers like Atomik and SABLE-CC.

However, it would be wrong to imply that Agentcities is simply an effort to deploy FIPA agents. The initiative aims to create a functional and useful network, and many aspects of this activity fall outside the scope of FIPA. Some examples:

- We wish to deploy a payment infrastructure on the network, and clearly the work done by web based payment schemes and XML authentication and security working groups will be fundamental.
- We wish to deploy ontology-servers on the network, and the efforts of the semantic web & description logic communities will inform our activity here.

In the rest of this paper we describe what an Agentcity is (section 2); the uses of an Agentcity (section 3); some of the components under development (section 4); the activities underway (section 5); how Agentcities is related to other

[1] ACKNOWLEDGMENTS: The research described in this paper is partly supported by the European Commission project Agentcities.RTD, reference IST-2000-28385. The author would like to thank the members of the various Agentcities initiatives around the world. The opinions reflected in this paper are those of the author and are not necessarily those of all the partners developing the Agentcities concept. The author would also like to thank Dr. Hamid Gharib, Dr. Robert Ghanea-Hercock & Dr Heather MacCann for their helpful comments on early drafts of this paper.

initiatives (section 6 & 7); some of the challenges Agentcities faces (section 8) and how to join the effort (section 9).

2 The Elements of an Agentcity

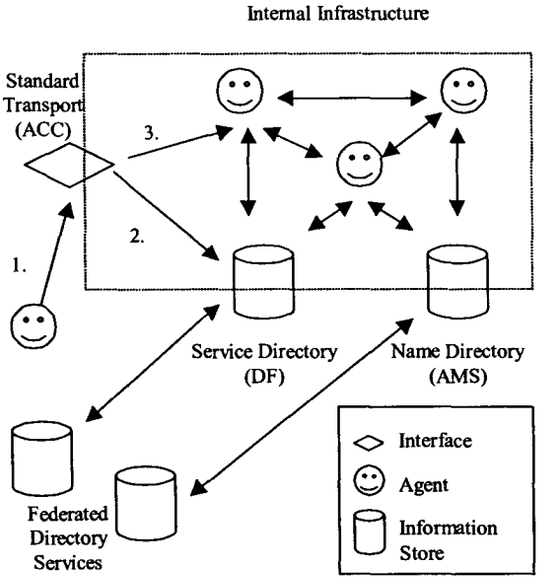


Figure 1. An Agentcity

Figure 1 shows the component parts of an Agentcity, based on the FIPA agent platform specifications (FIPA-00087, 00027). An agent can contact the Agentcity via the ACC (Agent Communication Channel) standard message transport (fig 1. 1). Currently FIPA-IIOP and HTTP transport interfaces are widely deployed on the network, however only HTTP has proved practically useful to this point. Other transports can be provided on the ACC and three transports that do not conform to FIPA standards have also been made available by various network users (these are Zeus-TCP/IP; AAP-MTP & FIPA-OS-RMI). It is expected that many others (eg. SOAP) will also be deployed and used, but in order to inter-operate with the rest of the network the ability to utilise FIPA-HTTP is required.

The ACC can forward messages to agents that are part of the Agentcity (fig 2. 2), or the message can be sent to the Agentcities directory services (fig 3. 3). There are two service directories: a FIPA-AMS (Agent Management Service) and a FIPA-DF (Directory Facilitator). The AMS directory provides a mechanism for namespace management in the Agentcity and the DF provides a way to register and locate services. The intended use of an Agentcity in an open environment requires that we provide a service ontology, by which we mean a descriptive categorisation, which can be used to reason about the applicability of a service to a task. This ontology will be based on DAML-S.

2.1 Agentcity Agent Attributes

Membership of an Agentcity has implications for the attributes and capabilities of an agent. Firstly the agent must be able to take part in conversations with other agents on the Agentcity. Secondly, Agentcities agents have a four layer transport model.

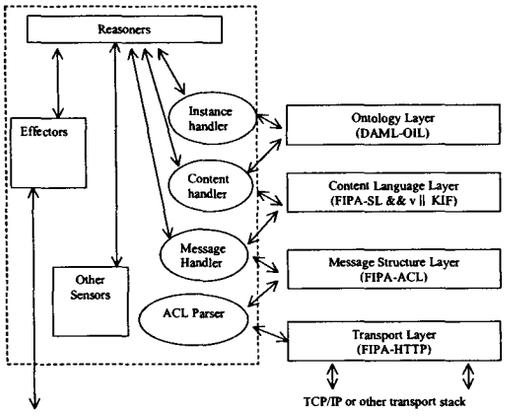
This model is shown in Figure 2 and described below:

- The agent must be able to receive messages on a FIPA transport channel (ACC). Currently HTTP is the default implementation, but in theory any message transport is admissible.
- The agent must be able to parse, process and generate FIPA-ACL messages
- The agent must be able to parse and process the content field of those messages which will contain content in one of the Agentcity content languages. Currently the two content languages chosen are FIPA-SL2 and KIF.
- Finally the agent must be able to make sense of the elements that it has parsed the content into: it must be able to map the tokens of the message to an ontology. We have chosen to use DAML-Oil as our ontology description language.

In Figure 2 a possible agent architecture is shown which deals with the message as it is parsed. Clearly information from each of the top three layers of parsing is used in the overall handling of the message - for example an "inform" message with content "((=(iota ?x (film_to_see ?x)) cinema :name ucg :type multiplex :time 20:00 :film die_hard))" - is required when deciding on an appropriate response. The agent needs to understand the message intent (inform), the type of information (a particular film time/place that is a film_to_see) that is being sent to it and the instance information in the content (a cinema film with a time in 24 hr format).

However, these requirements do not imply that an Agentcities agent must posses "competent" reasoning ability for the content language that it utilises. For example, it is unnecessary for an agent that advertises data access and storage services to be able to answer questions of logical inference about the data items it stores. The agent must be able to perform the tasks that it advertises, and address messages within the scope of those services. Other messages can simply be dealt with using the FIPA performative: "not-understood".

The other attributes an agent needs to be considered part of an Agentcity is the ability to register itself and the services that it can provide with the



Agentcities directories. To do this the agent must be able to represent the services that it can perform in the Agentcities service description format.

Figure 2. Agentcities message model and possible handling architecture

The FIPA service language & DF spec uses service identifiers to tag a service. Using languages like DAML-S should allow knowledge about the service to be registered with the DF. This service knowledge should allow other agents to plan the utilisation of the service as part of composite offerings.

3 Applications and Scenarios

The features of any network are motivated by its application. Agentcities is no exception; the network is aimed at decentralised business applications in which a number of organisations and individuals collaborate to generate some economic good. The trial scenarios under development in the .RTD project are:

- a group evening organiser, which is an application that will allow individuals to arrange an evening out using the current state of restaurant bookings, cinema film availability & reviews and transport solutions to decide on what to do.
- An event organiser that will allow an event to be organised and advertised using the network resources. This will include checking venue suitability, ticketing & catering.

We envisage these two scenarios joining together as the events that are supported by the network are used as options by evening organiser agents. For example, a music festival may be organised over the network and advertised as an entertainment service. An evening organiser agent may locate it, and offer it to the group as one part of an evening out.

In order to implement this sort of application the network must have the following properties:

- It must be accessible to providers - SME's must have a mechanism that they can use to create a presence on the network.
- It must be accessible to users - it must be easy to create information flows from the network to users.
- There must be mechanisms for paying for services securely.
- There must be mechanisms for agreeing the price of services.
- There must be mechanisms for integrating the information from various services together.

4 Component Services

In order to address the application level requirements that we have identified, a number of other services need to be available in the Agentcities network. Some of these are currently under development in the .RTD project:

- A secure payment system that will allow agents to exchange money (or tokens) for service execution. The service will also offer interfaces that will allow user accounts to be managed.
- An ontology service that will allow for ontologies to be shared between agents & agent developers, providing facilities for storage, retrieval and versioning. Later implementations of the service may provide ontology based reasoning and query services
- A distributed market place that provides a forum for the location of auctions and information on running auctions as well as auction execution.
- SME-Access – a portal that will permit users to customise and launch agents using a web based interface without the need for them to host an agent platform on a server of their own

5 Agentcities Activities

The Agentcities initiative has spawned a host of concrete projects which are now underway.

- Agentcities.rtd; a research activity funded under the EU Framework-5 program that has the objective of bootstrapping the Agentcities network.
- Agentcities.net; a network of excellence, again funded by EU. The network has funds to provide 30 grants of €10,000; a prize of up to €25,000; 70 grants of 6 month duration of €750 for student exchange and a support for a set of working groups. The network will disseminate

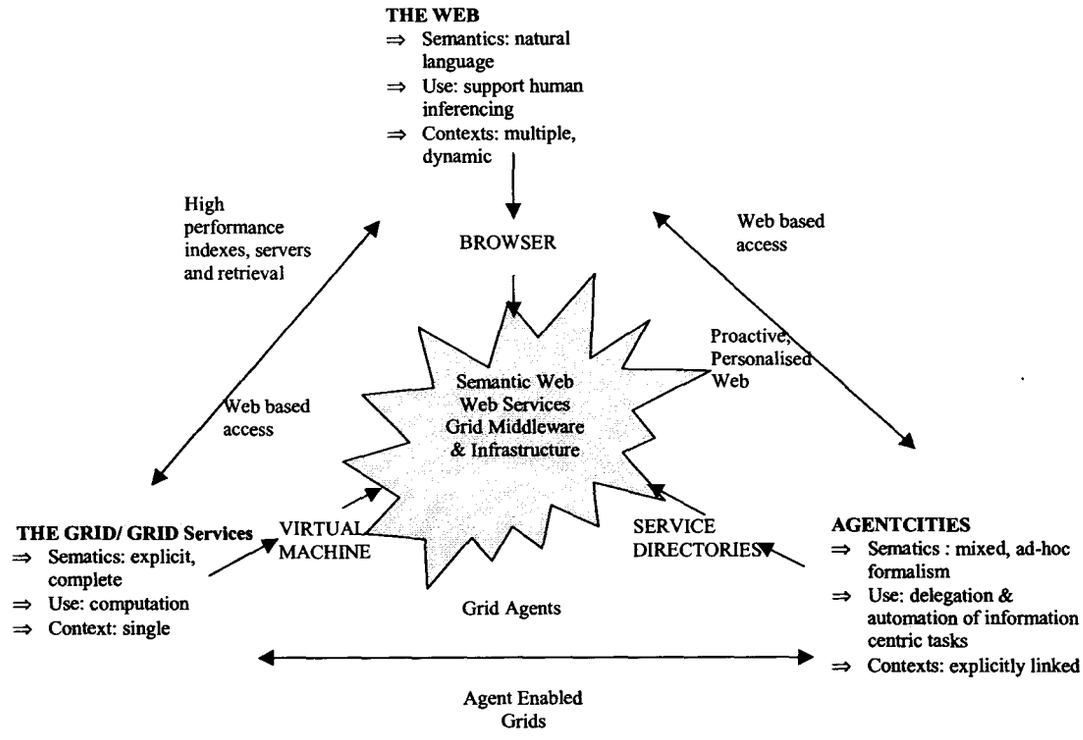


Figure 3. Relationships and uses for present and future information spaces and technologies.

and test the results of the .rtd project and will provide support for investigators in the EU who are not part of the .rtd and wish to contribute to and influence the networks development.

- Agentcities Australia; a research project funded by the state government of Victoria in Australia which is investigating the development of flexible interaction protocols between agents in Agentcities.
- Agentcities Finland; which is developing links to wireless platforms from Agentcities
- Agentcities Canada; which aims to support the deployment of a Canada wide agent network.

In addition to these funded and running projects, consortiums have been formed in many other countries including Mexico, USA, France and Spain to co-ordinate national efforts and to bid for support from funding bodies.

To manage the interaction between the various independently funded projects an independent open forum has been established – "The Agentcities Taskforce" (ACTF). The ACTF will act to co-ordinate the various Agentcities activities, and as a focus for dissemination of the initiative's achievements. The other function of the ACTF will be to provide a single point of contact through which Agentcities can interact with international standards bodies such as the Grid forum.

Section 9 of this paper gives information on how to get involved with these activities.

6 Relationship to other Initiatives

6.1 The Semantic Web

The Semantic Web initiative (Berners-Lee '98) is an attempt to provide the standards and technology that is required to add semantics to the world wide web. The objective is to provide mechanisms that allow competent expression of information in a machine processable form. That is to say to allow machines to perform inferences over the knowledge, data and information sources that are deployed on the WWW using variants of the HTTP protocol (such as HTTPS and HTTPR).

The activities of the Semantic Web community have led to the development of the DAML markup language, a dialect of RDFS which is implemented in XML. Many variants of the DAML markup

language are under development by many diverse projects and initiatives, including DAML-OIL (a description logic based ontology description language), DAML-S (a top ontology for services) and many others. Recently the W3C has started to discuss the development of OWL (Ontology Web Language) which is seen as a standardisation of DAML-OIL (although there appears to be some debate about this).

Essentially the Semantic Web is evolving into a set of standards for representing information and the things that can be done with information, such as querying and advertising.

6.2 Web Services

Web Services are a collection of tools and interfaces that enable the interoperation of separately developed corporate information systems such as e-procurement and e-sourcing systems. The .Net initiative is the Microsoft developed proprietary equivalent of web services. The component parts of a web service system are

- A UDDI server for the sharing of names and service access ports.
- WSDL interfaces that describe the calling interface to the programs & business objects that have been deployed. WSDL provides an XML encoded equivalent to the OMG's IDL which describes the parameters and return types of methods that can be invoked on the remote object.
- SOAP method invocation transports; SOAP is a mechanism for performing remote method invocation over the HTTP transport.

Other elements of a web services type system include protocols and methods for trust, authentication and security, and protocols and methods for login and session management.

7 Agentcities vs. Computational Grids

A Computational Grid can be defined as "a hardware and software infrastructure capable of providing dependable, consistent and inexpensive access to high end computing resources"(Foster & Kessleman 1998). It is interesting to contrast the approaches of the Agentcities initiative and the Grid community:

- Grid development emphasises *computation* and *storage*, Agentcities emphasises *delegation, inference* and *belief.*

- Grid computing requires new middleware and communications systems with enhanced performance; Agentcities relies on existing middleware and assumes current levels of performance.

- Globus (Foster & Kesselman 97) & Legion (Lewis & Grimshaw 96) enabled Grid's present resources as a single virtual machine or unified data model; "the grid is your peer", while Agentcities provides service, peer location and communication infrastructure, but no unified information model. Other "Grid" software include as Pastry (Rowstron & Drushel 2001) and OceanStore (Kubiatowicz et-al 2000)

While the Agentcities approach and the Grid approach are somewhat different it is clear that both systems will share a mutual environment: the world of the Semantic Web and Web Services. In addition to this the high performance computing emphasis of grid research is generating a new generation of high capacity, high performance networks. An infrastructure of computing surfaces and stores will obviously be useful in the context of web based services and in the context of Agentcities style agent systems. For example, Grid infrastructure could be used for providing extensions or equivalents to Google's web page index that is now over 2,000,000,000 pages in size. Currently Google type indexes must be run on a Linux cluster of over 10,000 servers (Google 2001). It is also likely that high performance indexing, computation and network search will expand the space of plausible Agentcities applications.

At the same time the aim of providing a virtual machine that represents the computational, programmatic and data resources required by a user implies that the instruments and resources in The Grid will need to be proactive and self organising to some degree. For example, a Biologist logging on to The Grid should find that his Grid VM provides optimised access to the human genome database, specialist DNA visualisation codes and a set of processors suited to his computation needs. Further more the instruments to be used should have taken and cached readings that will be required in the user session during slack time over night. She or he should not need to be aware of the process that underpins this, and should not be banned or prevented from accessing other resources, it is just that the most likely needs of the user should be the needs best addressed by the system as a whole. This sort of behaviour is often seen as the domain of agent technology.

Figure 3 represents one view of the relationship of the network computing initiatives. The use cases of "The Web", "The Grid" and "Agentcities" are qualitatively different; Web based systems support human "in the head" style reasoning; to us Grid based systems support computation and Agentcities

support delegated reasoning. We believe that the Web Services and Grid infrastructures that are under development will be useful to all three types of interaction. We also believe that the semantic markup and reasoning standards and systems which will become the Semantic Web will also be crucial to all three. Other views are proposed as part of the Open Grid Services Architecture and in De-Roure et-al 2001.

In any case, the relationship of the various communities developing these initiatives will always be shifting, and will always be subject to the vagaries of fashion and politics within the academic and commercial worlds. One thing is clear; there is a great deal that can be usefully learned and shared, by each community by engaging with and monitoring the activities of the others.

8 Open Questions and the way Forward

Several interesting questions have already been posed by the development and deployment of the Agentcities network.

8.1 The institution of an Agentcity.

Initially each node in the network has deployed a single agent platform for its geographic location, but it has rapidly become obvious that some locations are likely to have a number of platforms that support them. The initial example of this is the London Agentcity, where Imperial College and Queen Mary's College University of London, have deployed two platforms that they plan to operate co-operatively as London1 and London2 to form the London Agentcity. However like all big cities London has a host of Universities and internet service providing companies and several of these, in addition to IC and QMUL, have joined the Agentcities network of excellence and expressed interest in deploying an Agentcity node.

Obviously in the early stages of the project this problem can be easily managed at a personal level by co-operation and collaboration. However, if we accept that there is not a workable system of network authority then it is obvious that a more sophisticated, de-facto or self organising system of managing the representation of an area is required. This may take the form of a subscription based mechanism, where platforms subscribe to one another and agree that they form part of the same institution and share information on a different basis from the way that they interact with the rest of the network. For example platforms that form an Agentcity may have DF's that are more willing to accept federated requests from one another than from DF's on the open network. Alternatively some sort of indexing and clustering service may be required to create and maintain the knowledge that services about London should be part of the London Agentcity.

8.2 Firewalls and Security

Until the Agentcities initiative very few deployments of FIPA-agent-platforms on the open internet had taken place. While there were a large number of in house agent networks in use, these were almost all, with a few notable exceptions like Imperial College's FIPA-NET, operated behind a firewall. Because of this, security & deployment management have not been explicitly considered by the agent community until recently.

A clear example of this is that there are no standard Agent port numbers in TCP-IP. MTP (part of AAP) uses 4549 as a standard number, and also explicitly considers firewall configuration and policy issues in its design (McCabe 2001), but while HTTP nominates port 80, the use of it as an agent message transport does not adopt any such convention. Currently the first five platforms (alphabetically) in the platform directory use ports 8080, 3194,1045,7778 & 1099. The result of this is that while a platform can be deployed behind a firewall, which has only one incoming port, all outgoing ports have to be left open in order to be assured that a message received can be replied to.

This is a security risk as it makes Agentcities machines tempting targets as zero-day servers for hackers; being able to contact a number of other machines on different ports makes setting up an illicit forwarding and redistribution network easier. It would also make it less easy to control and catch the spread of Trojans over the network.

As the network develops, this issue will have to be resolved. It is probable that platforms will wish to have multiple in ports and therefore out ports as well; perhaps a range from 8000 to 8025 should be adopted as admissible port numbers to allow developers to deploy multiple ACC's on one platform or multiple platforms on one machine. The potential to deploy multiple ACC's is especially important if we are to build a scaleable network; current e-commerce sites expect peak traffic in the region of 60 hits per second (Harkins 2001). It is certain that at least these volumes will be required if e-commerce operations are to be profitable, and given the nature of agent systems it seems likely that messaging volumes in agent to agent applications will be much higher still, and that, for future applications, the messaging volumes will be exponentially higher.

9 Getting involved

9.1 Connecting to the Network

A simple way of getting started is to download one of the open source FIPA compliant agent platforms that are available.

Currently the following open source platforms are being used on the network, as are several proprietary platforms.

- AAP
  http://sourceforge.net/projects/networkagent
- FIPA-OS,
  http://fipa-os.sourceforge.net
- Leap/JADE
  http://sharon.cselt.it/projects/jade/
- Zeus
  http://www.btexact.com/projects/agents/zeus

Once a platform has been obtained you will need to configure your firewall to accept requests on the port that you have chosen and to permit outgoing requests to other agents. The platform can then be registered on the Agentcities web site in order for the meta directory agents to include your platform in their search.

The Agentcities Task Force has issued a document that describes in detail how to create an Agentcity and how to connect it to the network (Constantinescu etal 2002). This describes in detail how a platform can be configured to become part of the network.

9.2 Other ways to be involved

A general mail list is run by the ACTF and is open to all to join at

- http://srv01.lausanne.agentcities.net/dotorg/listinfo/discussion

Agentcities.NET is open to applications by all EU based organisations and runs a number of working groups and information days that are open to participation by all.

10 Summary

Agentcities is a project that is deploying FIPA agent technology in an open field trial. Applications are being developed that emphasise the unique attributes of agent technology, and as a result several interesting questions about the structure of the network and the systems that are needed to utilise it have arisen.

This paper has described the progress made in the short period of time that the network has been running, and the directions that the work is taking at this time.

Agentcities is an open initiative, and this paper is intended as an invitation to the reader to participate in creating a world-wide open environment in which autonomous, adaptive agents can be deployed.

11 References

Berners-Lee, T. "A Roadmap to the Semantic Web", http://www.w3.org/DesignIssues/Semantic.html, September 1998.

Constantinescu, I., Dale, J. & Willmott, S. "Connectiong to the Agentcities Network Recommendation.", Agentcities Task Force Recommendation Document actf-rec-00002 http://www.agentcities.org/rec/00002/actf-rec-00002a.pdf

De Roure, D., Jennings, N., & Shadbolt, N. "Research Agenda for the Semantic Grid: A Future e-Science Infrastructure", http://www.semanticgrid.org/v1.9/semgrid.pdf

Fikes, R. & McGuinness, D. L. An Axiomatic Semantics for RDF, RDF Schema, and DAML+OIL. Knowledge Systems Laboratory, January, 2001.

Foster, I. and Kesselman, C. (1997) "Globus: A Metacomputing Infrastructure", *International Journal of Supercomputing Applications*, 11(2): 115-128, 1997.

Foster,I and Kesselman, C. (1998) (eds) "The Grid: Blueprint for a New Computing Infrastructure", Morgan Kaufman Publishers, 550pp.

Google (2002) "Tech Highlights", http://www.google.com/press/highlights

Greaves, M.T. (2001) "Communication and Conversation in FIPA Agents" fipa.umbc.edu/18/greaves.pdf

Harkins, P. (2001) "Building a Large-scale E-commerce Site with Apache and mod_perl", Perl.Com, O'Reilly Newsletter, http://perl.com/pub/a/2001/10/17/etoys.html

Kubiatowicz, J., Bindel, D., Chen, Y., Czerwinski,S. Eaton, P. Geels, D., Gummadi, R., Rhea, S., Weatherspoon, H., Weimer, W. Wells, W. and Zhao,B. OceanStore: An Architecture for Global-Scale Persistent Storage , Appears in *Proceedings of the Ninth international Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000),* November 2000.

Lewis, M.J. & Grimshaw, A. (1996) "The Core Legion Object Model", *Proceedings of the Fifth IEEE International Symposium on High*

Performatince Distributed Computing, IEEE Computer Society Press, Lost Alamitos, California, August 1996.

McCabe, F (2001) "InterAgent Communications Reference Manual", http://sourceforge.net/docman/display_doc.php?docid=3288&group_id=3173

Rowstron, A & Druschel, P. "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems". IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany, pages 329-350, November, 2001

Willmott, S. N., Dale, J., Burg, B., Charlton, C. and O'Brien, P. (2001) "Agentcities: A Worldwide Open Agent Network", Agentlink News,8, pp13--15, November
http://www.AgentLink.org/newsletter/8/AL-8.pdf

jfipa - an Architecture for Agent-based Grid Computing

Amund Tveit
Department of Computer and Information Science, NTNU
IDI/NTNU, N-7491 Trondheim, Norway
amund.tveit@idi.ntnu.no
http://www.jfipa.org/amund/

Abstract

With the increasing focus on grid development, there is a need for proper abstractions for modelling grid applications. Viewed from a distributed AI perspective the most suitable abstraction is the concept of agents. In this paper an agent-based architecture for grid computing is considered. The architecture enables routing and handling of FIPA ACL messages.

1 Introduction

With the increasing importance and potential of the Internet as an efficient global communication and computing infrastructure, the development and popularity of concepts and technologies related to *grid computing* has accelerated. The grid has been described as *"coordinated resource sharing in dynamic, multi-institutional virtual organizations"* (Foster et al. (2001)). These virtual organizations (VO) and their members are interconnected by a network, typically the Internet. Some examples of resource sharing are cpu-intensive processing (e.g. physical simulations), data storage and online services

If grid computing is considered from a *Distributed Artificial Intelligence* (DAI) perspective (Moulin and Draa (1996)), it is clearly not a type of *Distributed Problem Solving* (DPS), since there is no or little central control in the grid. Its resemblance with a *Multi-Agent System* (MAS) is much higher, since entities in a MAS and the grid have autonomous behavior (i.e. distributed control).

Autonomy is one of the key abstraction features of agents (Wooldridge and Jennings (1995)). Other features of agents relevant for grid entities include *social ability*, as well as *reactive* and *pro-active* intelligence. This justifies the selection of agents as the main abstraction for grid entities.

1.1 Grid Routing Issues

However, since one often talks about *the* grid, it means that the *whole* Internet and the VOs can be seen as the edges and nodes of a graph representing the grid. This graph is *very* large, and far from being complete, which means that a message sent between two nodes need to travel a path through intermediary nodes before arriving at its final destination. The process of finding this path is called a *routing algorithm.*

Three important metrics of an efficient routing algorithm are the number of intermediate nodes in a path, the path cost, and the delay (Tel (2000)). They should all be minimized according to expression (1).

$$min\ (c_1 \cdot hops + c_2 \cdot pathcost + c_3 \cdot delay) \quad (1)$$

When messages are forwarded through the grid based on other criterias than the receiver address, e.g. service description or price, the process is commonly called *automatic brokering*. Even though these processes have new metrics dependent on the query (e.g. quality, cost or relevance of the result), the fundamental metrics of routing are still of importance.

1.2 Problem

The overall problem is: *how to enable efficient routing and brokering in an agent-based grid?*

More particularly, how to create an architecture: *1) that is pluggable with respect to routing algorithms, 2) has a simple and easy-to-use API, 3) has high processing and IO performance, and 4) has a robust implementation.*

The rest of this paper is organized as follows. Section 2 describes and discusses the architecture, section 3 describes the implementation, section 4 describes related work, and finally the conclusion with future work.

2 Architecture

2.1 Architectural Goals

The overall goal of the jfipa architecture is to become a simple, yet efficient and extensible, router architecture supporting agent-based grid computing. Agents should be able communicate using the speech act-based (Searle (1969)) FIPA Agent Communication Language (ACL). The primary supported encoding of the ACL should be XML, and the primary communication protocol (application-level) should be HTTP.

As the j in jfipa insinuates, java is the primary implementation language, but also other common programming languages will be able to use jfipa through a planned XML-RPC interface.

2.2 Scalability Goals and Approach

The jfipa architecture should be able to scale in several different dimensions: 1) increase in the agent identification address space (Postel and Touch (1999)) (related to future IPv6-based Internet), and 2) an increase in total number of agents (Rana and Stout (2000)) (residing on different jfipa-nodes) and messages. The primary approach of enabling this scalability is through efficient routing of ACL messages by supporting pluggable routing algorithms. Other scalability-enabling approaches (e.g. caching) is outside the scope of jfipa.

2.3 jfipa Overview

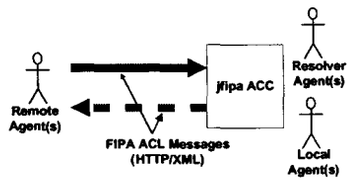


Figure 1: jfipa Overview

Figure 1 shows the three main components of the jfipa architecture. *Resolver agents* provide routing and brokering support for incoming ACL messages, and the *local agents* provide any type of service that is not related to routing or brokering. The *jfipa Agent Communication Channel* (ACC) provides support for communication, parsing and queueing of ACL messages, as well as administration of the resolver and local agents. *Remote agent(s)* are agents that access the jfipa architecture externally using FIPA ACL.

Note that the term *local agent* doesn't necessarily mean that the agent and the jfipa ACC reside on the same computer with the same network address, but it does mean that the local agents use the services of the communication channel, hence it doesn't need to have its own FIPA ACL handling mechanisms.

2.4 jfipa ACC Architecture

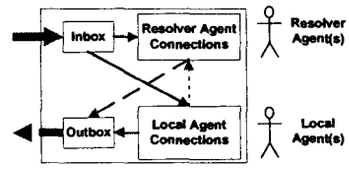


Figure 2: Agent Communication Channel

Figure 2 shows the four main components of the jfipa ACC. The *Inbox* is where incoming ACL messages are received (i.e. from remote agents), *Outbox* is where the internally generated or routed messages are being sent from (i.e. messages from local or resolver agents). *Resolver Agent Connections* and *Local Agent Connections* handles ACC communication for the resolver and local agents, respectively.

The reason for having the Inbox and the Outbox handling communications instead of the Resolver and Local agents themselves, is that logging and administration becomes simpler.

2.5 Inbox

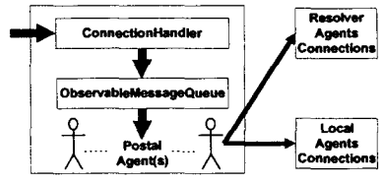


Figure 3: ACC Inbox

ConnectionHandler handles incoming network requests and inserts them into the *ObservableMessageQueue*, which notifies a *Postal Agent* about the new message. Connection between the ObservableMessageQueue and the Postal Agents are based on the *Observer Design Pattern* (Gamma et al. (1995)).

2.6 Postal Agent Architecture

The postal agent extracts and parses the HTTP message its FIPA ACL XML-encoded envelope and message. It continues by checking if the message is addressed to a

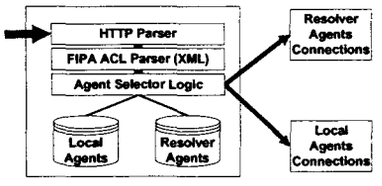


Figure 4: Postal Agent Architecture

particular *local agent*, otherwise it passes on the message to the *resolver agents* for potential routing or brokering.

The Postal agents also handles the repository of registed local and resolver agents.

2.7 Agent Connections Architecture

The *Resolver Agents Connections* and *Local Agents Connections* architectures are very similar, except that the latter also supports forwarding of messages to the prior based on the results from local agent processing. Support for creating a large number[1] of (sophisticated) local agents is outside the scope of jfipa, and is supposed to be handled by other agent platforms that only use jfipa for external Fipa-based interaction (e.g. Agora Matskin et al. (2000)). Connections between jfipa and the other agent platform is done using two observer design patterns, where the local agent is the observer of the Incoming Messagehandler's queue and the observable for the Outgoing Messagehandler.

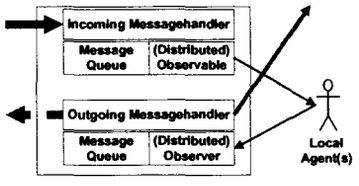


Figure 5: Agent Connections Architecture

In order to support interactions between few (non-sophisticated) local agents (e.g. wrapper agents for existing legacy systems), jfipa will use a hashtable of incoming/outgoing message queues where each hash key represents the name of a local agent, this ensures efficient ($O(1)$) sending of messages. The receiving agent gets notified about messages by using observing its message queue.

2.8 Routing Issues

The main data structures for a routing algorithm are *routing tables*. Routing tables have information about the network topology, or in other words, knowledge about which direction (peer/neighbour) to best send/forward messages.

If the network is small (few nodes and edges) and relatively static (slow changes in topology), routing tables can efficiently being calculated using shortest path algorithms, e.g Bellman Ford's (BF) or Dijkstra's algorithm (Cormen et al. (1990)).

The main disadvantage of the shortest path algorithms is that they don't scale well with rapid increase in network size and changes in topology. They tend to be suboptimal because they are sending *all* traffic to the same destination through a *single* path (i.e. using only one peer per destination), instead of dividing the traffic among *several* paths (David H. Wolpert and Turner (2000))

The application of machine learning algorithms in adaptive routing, in particular reinforcement learning, has been shown to improve network throughput by up to three and a half times than routing with the BF algorithm (David H. Wolpert and Turner (2000)). The main reasons for the improved performance is that the reinforcement algorithms adapt to the traffic by being less bound to only one path per destination than the case for BF. The percentage of which peers should forward which messages to a particular receiver is stored in a proportion vector $\vec{p}$ ($\Sigma_i p_i = 1$, if the BF algorithm had such a vector it would store a 1 in precisely one of the positions in the vector)

These promising results by the applying machine learning for efficient routing motivates the support for pluggable routing algorithms in the architecture, as shown in figure 6.

2.9 Resolver Agent Architecture

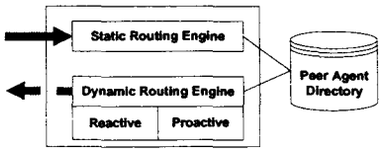


Figure 6: Resolver Agent Architecture

Static Routing Engine supports simple routing based on routing table lookups.

Dynamic Routing Engine supports two types of routing. The first is the *Reactive* router, it routes or prunes the current incoming message (i.e. stimulus-response on incoming messages).

The second is the *Proactive* router, it tries do larger changes such as altering topology based on clustering-ideas in excess cpu periods. The motivation behind this is to try to reduce the number of network hops for ACL messages by putting semantically similar agents closer. In

[1] more than 100 agents

the case of a recommender system, this could be done by calculating the proximity between every pair of peers of resolver agents (based on historic messages to/from the peer) and notify all pair of peers that has a high resemblance (Tveit (2001b)).

Peer Agent Directory is the routing table, with information about other jfipa nodes and agents.

Other tasks of the proactive router could be to try to estimate changes in topology based on analysis of messages (frequency, resolver information, content etc.), in order to optimize the routing tables. Interaction detection is another possible task, e.g. figuring out when there is an auction and which type of auction etc. The proactive router could also do statistical calculations and estimates in order to try to improve performance.

Interactions with other resolver agents related to feedback on routing decisions is either done using the inbox/outbox FIPA-ACL based network IO mechanisms (possibly with the development of routing ontologies), or by proprietary network IO in the Dynamic Routing Engine.

3 Implementation

The implementation of jfipa is still work in progress, so far a set of efficient parsers (for HTTP, XML-encoded FIPA Envelopes and Messages) have been made, as well as the overall architectural choices.

3.1 jfipa benchmark

Figure 7 compares jfipa parsing performance with state-of-the art XML parsers. Crimson is Sun Microsystems own XML parser, now taken over by the Apache project, and Xerces is being described as "the next generation, high-performance XML parser". Both were clearly outperformed by jfipa[2].

Each measurement point is the average of 10000 parsing rounds of a FIPA XML Envelope, this is done in order to make the parsers initialization costs neglible. A spreedsheet containing measurements and test-files can be found at jfipa.org/publications/2002/jfipaBenchmark.xls.

One of the main reasons for the performance differences performance is that Crimson and Xerces need to build an memory model at runtime of the XML DTD in order to validate the XML document (XML-encoded FIPA Envelope in the test case). jfipa allready has this model pre-runtime since it is specialized towards handling XML-encoded FIPA ACL Envelopes and Messages. Other optimization approaches in jfipa is efficient reuse of allocated objects, avoidance of slow String-based operations, and buffered IO.

[2]Measurements were done on a Dell Inspiron 4100 w/1GHz PIII CPU, 512MB ram, W2K

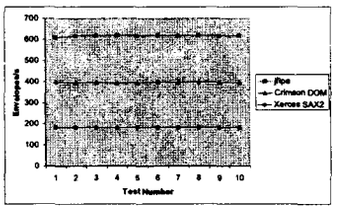


Figure 7: jfipa XML Parsing performance

3.2 Code structure

The code structure of jfipa is divided in three main packages:

1. api
2. implementation
3. test

The test package is testing the api methods (i.e. interfaces), but with the selected implementation. The motivation behind this structure is that it should be easy to test new implementations with existing test code, e.g. porting the implementation package to support java for mobile devices (J2ME).

3.3 Code quality

In order to ensure that the jfipa code is fairly solid, unit tests are created, this allows quick retesting of the *whole* system each time changes are made.

3.4 Open Source Licence

jfipa is under the MIT Open Source licence. As opposed to the Gnu Public Licence (GPL) and the Lesser Gnu Public Licence (LGPL), the MIT open source licence is rather non-problematic regarding commercial utilization of the software.

3.5 jfipa Code Availability

The jfipa source code is being hosted by VA Software's SourceForge.net on http://jfipa.sf.net/. For documentation and examples, the jfipa homepage - http://www.jfipa.org is the place to look.

4 Related Work

The architectures that resemble jfipa most, is JATLite (Jeon et al. (2000)) and JATLite ACL. JATLite supports messages of the Knowledge Query and Manipulation Language (KQML), and JATLite the messages with the Lisp-

syntax of FIPA ACL. Lately there seems to have been little development on JATLite, and the code doesn't take advantage of newer java language opportunities.

The SoFAR multi-agent framework et al. (2000) is designed for distributed information management tasks in a grid environment, it operates on a higher abstraction level than the message-oriented JATLite and jfipa, since it is centered around agent services and is geared towards human users and not only agents.

There also exists several other FIPA-based multi-agent frameworks (e.g. FIPA-OS Poslad et al. (2000) and Jade Bellifemine et al. (2000)), but they are much more comprehensive since they seem to try to support the *whole* FIPA standard, and not as geared towards routing and brokering mechanisms as jfipa.

5 Conclusion

In this paper an architecture for agent-based grid computing has been presented. The main contribution is the architecture itself, as well the (in-progress) open source implementation of it.

The jfipa architecture is pluggable with respect to routing algorithms, its API is evolving towards becoming simple to use (the current jfipa implementation is a refactoring of the first implementation with emphasis on simplifying API), the (parsing) processing performance is high, and the implementation is fairly robust mainly due to the many unit tests applied.

Ongoing and future work include finishing the implemention the architecture and apply it in large-scale grid experiments of data mining in massive multiplayer computer games. The primary research problem will be knowledge discovery of player logoff patterns in order to get 1) a player stability metric for massive multiplayer games, and 2) being able to do counter-actions, e.g. automatic recommendations based on collaborative filtering in order to keep players longer (and generating more revenue for the game provider). Integration of jfipa software with Agent-Oriented Software Engineering methodologies is also a possible direction, we refer to Tveit (2001a) for a overview of such methodologies.

Other areas considered investigating is the simulation of financial products (e.g. derivatives) related to the upcoming market of carbon dioxide quoate trading (see PointCarbon.com). jfipa will eventually be integrated with the Agora Multi-Agent framework (Matskin et al. (2000)), in order to make Agora support FIPA ACL communication.

Acknowledgements

I would like to thank my Agentus colleague Thomas Brox Røst for fruitful discussions on java performance, as well as my supervisor Mihhail Matskin. This work is supported by the Norwegian Research Council in the framework of the Distributed Information Technology Systems (DITS) program and the (ElComAg) project.

References

Fabio Bellifemine, Agostino Poggi, and Giovanni Rimassa. Jade - a fipa-compliant agent framework. In *Proceedings of the 5th International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM)*, pages 97–108, 2000.

Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.

Chris J. Merz David H. Wolpert, Sergey Kirshner and Kagan Turner. Adaptivity in agent-based routing for data networks. In *Proceedings of the Fourth International Conference on Autonomous Agents*, pages 396–403. ACM, june 2000.

Luc Moreau et al. SoFAR with DIM Agents. In Jeffrey Bradshaw and Geoff Arnold, editors, *Proceedings of the 5th International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM 2000)*, pages 369–388, Manchester, UK, 2000. The Practical Application Company Ltd.

Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the Grid: Enabling scalable virtual organization. *The International Journal of High Performance Computing Applications*, 15(3):200–222, Fall 2001. ISSN 1094-3420.

Erich Gamma, Richard Helm, Ralph Johson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

Heechol Jeon, Charles Petrie, and Mark R. Cutkosky. Jatlite: A java agent infrastructure with message routing. *IEEE Internet Computing*, pages 87–96, March–April 2000.

Mihhail Matskin, Ole Jørgen Kirkeluten, Svein Bjarte Krossnes, and Øystein Sæle. Agora: An infrastructure for cooperative work support in multi-agent systems. In Tom Wagner and Omer F. Rana, editors, *Infrastructure for Agents, Multi-Agents and Scalable Multi-Agent Systems, Lecture Notes in Computer Science*, volume 1887, pages 28–40. Springer-Verlag, 2000.

Bernhard Moulin and Brahim Chaib Draa. An overview of distributed artificial intelligence. In Greg M. P. O'Hare and Nicholas R. Jennings, editors, *Fundamentals of Distributed Artificial Intelligence*, pages 3–56. John Wiley and Sons, 1996.

Stephan Poslad, Phil Buckle, and Robert Hadingham. Open source, standards and scalable agencies. In *Proceedings of the fourth International Conference on Intelligent Agents*. ACM, 2000.

Jon Postel and Joe Touch. Network infrastructure. In Ian Foster and Carl Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure*, chapter 21, pages 552–562. Morgan Kaufmann Publishers, Inc., 1999.

Omer F. Rana and Kate Stout. What is scalability in multi-agent systems? In *Proceedings of the fourth International Conference on Intelligent Agents*, pages 56–63. ACM, june 2000.

John R. Searle. *Speech acts: An Essay in the Philosophy of Language*. Cambridge University Press, 1969.

Gerard Tel. *Introduction to Distributed Algorithms*, chapter 4, pages 103–154. Cambridge University Press, 2 edition, 2000.

Amund Tveit. A Survey of Agent-Oriented Software Engineering. Proceedings of the First NTNU CSGS Conference, http://www.jfipa.org/publications/AOSE/, May 2001a.

Amund Tveit. Peer-to-peer based recommendations for mobile commerce. In *Proceedings of the First International Workshop on Mobile Commerce*, pages 26–29, Rome, Italy, July 2001b. ACM.

Michael J. Wooldridge and Nicholas R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 2(10):115–152, 1995.

AISB Symposium on

AI and Grid Computing

Recent advances in computer science facilitate a paradigm shift towards large scale computing – the Grid, with 'parallel and distributed' computing playing an important role. The Grid paradigm offers an important abstraction for combining national and continent wide computational resources, to enable application scientists and domain experts to work more effectively, and, subsequently, to do better science. Driven by the unprecedented increase in Internet usage, Grid-oriented computing will provide the next phase in the way we manage and disseminate knowledge. The Grid can be seen as a mechanism for integrating resources, which can range in complexity from dedicated parallel machines (maintained by national centres) to low-end clusters of workstations, connected by Beowulf/Linux, for instance. Managing such diverse computational platforms has been an active research area, and systems such as GLOBUS (Argonne) and Legion (Virginia) are often quoted as core infrastructure for managing Grid services. An alternative perspective, which subsumes the resource management aspect, involves viewing Grids as an integration of information services, which range from application specific data repositories, to data mining tools that can support the extraction of useful knowledge from such data.

Activity in Grid computing has also accelerated recently as a result of advances in information technology and its use, such as:

- Component based software development
- High speed networks
- Standardisation of interfaces to databases and data repositories
- Virtual machines and cluster computing
- Public domain and community software licensing arrangements
- On-demand (on-use) software payment schemes
- Network aware interfaces and visualisation

To make effective utilisation of resources across a Grid that spans organisational boundaries, it is imperative that the underlying infrastructure support intelligence. Intelligent software is required to undertake resource and service management, service discovery, service aggregation/decomposition, and support performance management. Commercial systems will also require the underlying infrastructure to respect site autonomy, and particular site specific policies on usage.

The objective of this symposium is to bring together researchers in computer science and AI, to discuss issues in managing Grid services and resources. Six papers are presented in this symposium, covering aspects of developing distribution mechanisms for computing, such as the paper by P. Mathieu, J.C. Routier, and Y. Secq on "RAGE". Another aspect discussed is middleware to connect Grid based applications, such as the paper by S. Newhouse et al. on ICENI,

Laying the Foundations for the Semantic Grid

Steven Newhouse, Anthony Mayer, Nathalie Furmento,
Stephen McGough, James Stanton and John Darlington
London e-Science Centre,
Imperial College of Science, Technology and Medicine,
180 Queen's Gate,
London SW7 2BZ, UK
icpc-sw@doc.ic.ac.uk
http://www-icpc.doc.ic.ac.uk/components/

Abstract

Information relating to the resources, applications and the user's wishes are key to the transparent and effective exploitation of the federated resources within Computational Grids. These federated data, computational or software resources are owned by real organisations and made available as services to different computational communities or virtual organisations spanning multiple administrative boundaries. Higher-level services use the information relating to the resources' capability and the mechanisms for service fulfilment to automatically discover interoperable services and select the most appropriate service for the user with minimal human intervention.

Within this paper we describe a service-oriented Grid architecture that utilises existing web service protocols to federate resources into computational communities. The service-oriented information contained in the computational communities is exploited by higher-level services to ensure effective utilisation of the resources by both its providers (the resource owners) and consumers (the users). We believe the systematic definition, presentation and exploitation of such information constitutes the first step towards the construction of a Semantic Grid.

1 Background

Computational Grids have provoked widespread interest within the scientific and engineering community over the last decade (1). There are now many global projects focussing on the development of Grid middleware and the mechanisms needed by the applied science community to effectively exploit these infrastructures (2). These projects are all being driven by the needs of a diverse applied science community involving both data and compute intensive applications (e.g. EU Datagrid, PPDG, Griphyn, NASA's Information Power Grid).

Heterogeneous resources within these Grids are forged into a single virtual organisation through the use of middleware. The Grid middleware collects and publishes information relating to the resources within the organisation while providing secure platform neutral interfaces to the underlying resources. Globus is one such widely deployed middleware that uses the Metacomputing Directory Service (MDS) to hold and distribute information through a hierarchical network of MDS server within a virtual organisation (3). Access to the resources within a virtual organisation is controlled through a 'gatekeeper' which verifies a user's identity through an X.509 public key certificate.

The specification, development and standardisation of Grid related protocols is taking place through the Global Grid Forum (http://www.gridforum.org/), formed through the merging of activities in North America, Europe and Asia. Its role, through the activity of its research and working groups in areas such as security, information management, applications, etc., is to provide a collaborative forum for researchers in industry, computer science and the applied science communities. Its meetings now attract several hundred researchers from around the world.

The activity within the Grid community can be compared to that within the Web community a decade ago. At that time there was an active worldwide research community developing competing and incompatible protocols, and innovative functionality within the server and client browsers. Coordination and standardisation of these activities was left by the community to the World Wide Web Consortium (W3C). Since 1994 it has guided the evolution of the Hyper-text Markup Language (HTML) and produced the several new standards such as the Extensible Markup Language (XML).

The recent emergence of business to business e-commerce has exposed the limited capabilities of existing web protocols when used to develop higher-level services. Currently, the division between page content and its presentation is frequently blurred within HTML encoded web pages. The W3C has been instrumental in developing approaches that enable a clear separation between the content (encoded as an XML schema) and its visual repre-

rescheduling as 'better' resources become available or computational steering of the application (6).

3 Grid Programming Model

It is impossible to proscribe a Grid programming language on the diverse applications within the e-science community. Likewise, it is difficult to proscribe a programming model unless it provides the flexibility to encompass existing legacy applications and those compatible with current software engineering practices.

We are prototyping a component based model that allows legacy applications to be encapsulated as a single component with defined interfaces (7; 8). Our model allows a component's interface to be matched to any number of valid implementations. The decoupling between a components interface and implementation allows the component to be deployed on the 'best' currently available execution platforms within a distributed Grid environment (9). For example, each component may have several implementations such as different algorithms (e.g. iterative or direct solvers) for different platforms (e.g. Solaris or Linux) for different architectures (e.g. serial or parallel). We describe the component interface, its implementations and capabilities through CXML - an XML schema for Component applications (10)

More generally, we define an application as a network of linked components where we are able to describe the frequency and data volume of their interactions. By combining our knowledge of these interactions with information as to how a component's implementation will behave on a particular platform, with data relating to the platforms current state, we are able to optimise its performance within a particular policy (5). The decoupling between an interface and its implementation also allows us, by storing persistent data outside of the component, to migrate an application's components to other resources during execution.

In our model, a user submits a job (as an application specification) to the 'Grid' by placing a CXML description of the application network and the user requirements into the public computational community. The application mapper matches the components used in the application to the implementations that exist within the resources in the computational community. The application network is instantiated on distributed resources with the best component implementations to maximise a user defined criteria. The effective deployment of the application across distributed resources is enabled by the rich metadata relating to the resources, application structure and implementations available to the higher-level services. A component expects to be deployed into a local 'Grid container' that provides access to a minimal set of basic services. The detailed definition of these container services is currently under investigation.

4 Grid Services

Our experiences with ICENI and other infrastructures, such as Globus, have demonstrated the need for a minimum set of services to support e-science within computational Grids. We characterise these services into two groups: low-level services that interface directly to the underlying Grid fabric or provide essential services and higher-level services utilising these lower-level services. All services are registered with a Registry Service in the local computational community that exchanges information with its peers in other organisations.

4.1 Registry Service

The Registry Service (like the Jini Lookup Service) provides a 'known' point for information exchange between clients and other services. The local private registry service within an organisation federates its internal services into a community wide infrastructure through the actions of the domain manager. The domain manager 'pushes' information relating to capability and usage policies of the local community's services to (potentially) several public community wide Registry Services within different virtual organisations.

The domain managers and community wide Registry Service can be viewed as part of a peer-to-peer network. We intend to use this structure to propagate the services within one community to other computational communities. Propagation continues while there is an intersection between the acceptable useage policies of services from the remote domain with the local user community. This approach should ensure that all users allowed to use a resource will have access to it providing there is at least an indirect link between the two communities. Modelling of this infromation structure and an examination of the service propagation distance is underway.

4.2 Low-level Services

The local private Registry Service will contain a number of low-level services:

- Service Register – provides a mechanism for service registration, discovery and instantiation
- Authentication – verification of the user or host using X.509 certificates
- Authorisation – expression of a resources' usage and access control policy
- Execution Resources – invoke a deployed component on a resource
- Software Resources – an index of the locally deployed software components

the Jini service discovery mechanisms. The ICENI services are made available to clients using a web service protocol through a proxy embedded within an application server. See Figure 2. Both the internal ICENI and SOAP encoded web services messages use the same XML schema to describe the message content.

6 The Semantic Grid

The Grid middleware now being developed to support e-Science within the UK and elsewhere is evolving into a service-oriented architecture built upon standard web protocols such as XML, SOAP, WSDL and UDDI. While developing protocols within such a framework promotes interoperability there is no certainty that these protocols will always be understood. It is here, we feel, that a Semantic Grid will start to develop as services become capable of discovery and self-organisation.

Tim Berners-Lee defines the Semantic Web as 'an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation.' (12). Likewise, the Semantic Grid can be described as an 'extension of the current Grid in which information and services are given well-defined meaning, better enabling computers and people to work in cooperation'. In such an environment it is essential that information relating to the needs of the user and their applications, and the resource providers and their networking, storage and computational resources all have easily discovered and defined meaning that can be used by higher-level services to effectively exploit the Grid.

The development of these service-oriented ontologies will enable compatible services to autonomously build the large complex distributed computing environment that constitutes the Grid. Fundamental to this goal is the expression of Grid services through well defined protocols. From these protocols we are able to understand the capabilities of the service and, potentially, reason as to how it can interact with other services to meet the needs of the users. This reasoning can take place at several levels, from service composition and application assembly to the representation and exploitation of knowledge generated through data and computation services (13). The mechanisms for exploiting the service and data ontologies within the Semantic Grid may range from simple schedulers, application mappers and resource brokers to sophisticated, autonomous, mobile and intelligent agents.

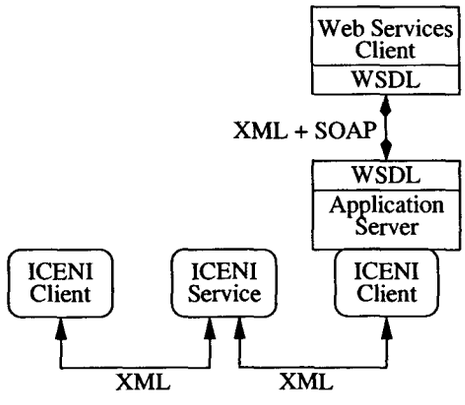


Figure 2: Extending ICENI to support interaction with web services.

7 Conclusions

Computational Grids are demonstrating a convergence of many existing areas of computer science research. The high performance computing community are developing new algorithms to support heterogeneous wide area computation between different supercomputers. Applications are having to deploy to new resources expecting only the services available within a 'standard container' environment. Users and applications are discovering distributed resources and services maintained through a network of peer-to-peer directory services. The autonomous discovery and assembly of a Grid environment from the available services is reducing the complexity involved in constructing a complex software environment while improving robustness through multiple services.

We consider the movement towards a Semantic Grid (at both the service and knowledge layers) as essential in simplifying the effective utilisation of sophisticated distributed services. The description of these services (using existing web-service protocols) will enable their intelligent composition and exploitation with minimal human interaction. The transparent and optimal delivery of sophisticated computational and data services to the applied science community will be key to the successful adoption of e-science.

We have shown how ICENI will continue to use Jini as a registration and service discovery mechanism while exposing its functionality within a web services framework to promote interoperability with other infrastructures. The XML schemas governing the interaction between different ICENI services will continue to be developed and encapsulated within SOAP for use by the web services interface. Services within our computational communities will be propagated to other organisations through a peer-to-peer mechanism.

Acknowledgements

Steven Newhouse gratefully acknowledge the formal and informal discussions that have taken place within the Architectural Task Force of the UK e-Science Core Programme which have helped clarify some if the issues discussed within this paper.

Multi-Agent Support for Internet-Scale Grid Management

B.J. Overeinder, N.J.E. Wijngaards, M. van Steen, and F.M.T. Brazier
Department of Computer Science, Faculty of Sciences, Vrije Universiteit Amsterdam,
de Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands
{bjo,niek,steen,frances}@cs.vu.nl

Abstract

Internet-scale computational grids are emerging from various research projects. Most notably are the US National Technology Grid and the European Data Grid projects. One specific problem in realizing wide-area distributed computing environments as proposed in these projects, is effective management of the vast amount of resources that are made available within the grid environment. This paper proposes an agent-based approach to resource management in grid environments, and describes an agent infrastructure that could be integrated with the grid middleware layer. This agent infrastructure provides support for mobile agents that is scalable in the number of agents and the number of resources.

1 Introduction

Computational grids are wide-area (Internet-scale) distributed environments that differ from conventional distributed computing by their focus on large-scale resource sharing, innovative applications, and high-performance orientation (Foster et al., 2001).

In a grid architecture, four levels of management can be distinguished: fabric, connectivity, single resource, and collective multiple resources. The fabric layer typically constitutes computational resources, storage resources, network resources, and code repositories. The connectivity layer deals with easy and secure communication by providing single sign on, delegation, integration with various local security solutions, and user-based trust relationships. The resource layer is concerned with individual resources, and the two primary classes of resource layer protocols are information protocols and management protocols. The collective multiple resources layer provides directory services, co-allocation, scheduling, and brokering services, monitoring and diagnostics services, data replication services, grid-enabled programming systems, workload management systems and collaboration frameworks (problem solving environments), etc.

In particular, coordinating collective resources is a complex high-level task that integrates the multiple resources into a wide-area distributed system. Many of the services in this layer can be effectively facilitated by applying *multi-agent systems*. Co-allocation, scheduling, and brokering services, monitoring and diagnostic services, workload management and collaboration frameworks, community authorization servers, community accounting and payment services, and collaborative services are processes that require intelligence, autonomy, and social capabilities: all qualities that are characteristic to intelligent agents (Wooldridge and Jennings, 1995).

A distinct problem is *scalability* in Internet-scale distributed systems like the Grid. In this paper, scalability refers to scalability of the wide-area distributed computing infrastructure and services, not of applications. Services such as resource management, co-allocation, and scheduling must deal with scalability problems that appear in large-scale, wide-area distributed systems (Wijngaards et al., 2002). Centralized (client-server) approaches have scalability problems as there is one central authority coordinating the activities. Hierarchical approaches already direct to a scalable solution, but peer-to-peer interaction strategies as embraced by agent technologies seems to be the most promising approach to provide scalable and adaptive services.

This paper presents a multi-agent infrastructure, called AgentScape, that can be employed for an agent-based approach to integrate and coordinate distributed resources in a computational grid environment. In particular, scalability, heterogeneity, and interoperability are discussed in perspective to a grid environment and the proposed multi-agent infrastructure.

2 Background

A number of initiatives to apply agents in computational grids have been initiated in recent years. Manola and Thompson (1999) present an overview of different perspectives to grid environments and describe DARPA's Control of Agent-Based Systems (CoABS) agent grid. In the CoABS Grid, a number of application level and functional requirements hold. Specifically, applications are considered to have multi-year lifetimes, evolving and changing requirements, are adaptable and scalable, and allows for system management without explicitly monitoring all components all the time. Practically, agent

technology is expected to help to provide more reliable, scalable, survivable, evolvable, adaptable systems, and help to solve data blizzard and information starvation problems. From a functional point of view, the CoABS Grid knows not only about agents, but also about their computational requirements, and about available computational (and other) resources. Hence, the CoABS Grid provides a unified, heterogeneous distributed computing environment in which computing resources are seamlessly linked.

Bradshaw et al. (2001) remark that "cyberspace" is currently a lonely, dangerous, and relatively impoverished environment for software agents. Consequently, most of today's agents are designed for "solitary, poor, nasty, brutish, and short" lives of narrow purpose in a relatively bounded and static computational world. They argue that focusing greater attention to making the environment in which agents operate more capable of sustaining various types of agents and collaboration groups, would simplify some of these problems. The CoABS Grid provides the infrastructure for large-scale integration of heterogeneous agent frameworks. The CoABS Grid capabilities have been extended by integrating the NOMADS agent environment for strong mobility and safe execution and the KAoS framework for policy-based management of agent domains to support long-lived agents and their communities (Bradshaw et al., 2001).

A good example of an agent grid is presented by Rana and Walker (2000). They identify the need to combine problem-specific problem solving environments (PSEs), facilitating interoperability between various tools and specialized algorithm each PSE supports. An agent based approach to integrate services and resources for establishing multi-disciplinary PSEs is described, in which specialized agents contain behavioral rules, and can modify these rules based on their interaction with other agents and with the environment in which they operate.

Another type of application of agents in distributed or parallel computing is typically with master-slave computations in wide-area distributed environments (Ghanea-Hercock et al., 1999). In these systems, large computations are initiated under control of a coordinating agent that distributes the computation over the available resources by sending mobile agents to these resources. In this perspective it is in some way similar to the Condor system or the SETI@home experiment, which also incorporate coordination and distributing the computation of the available resources. Essentially, the added value of the distributed computing agent systems is similar to the agent grid: coordination and seamless integration of the available distributed resources.

Principal idea behind a grid infrastructure is resource sharing and providing services. The introduction already outlined the four levels of management that can be determined in the process of sharing resources over a wide-area network. With respect to providing services in a grid environment, Foster et al. (2002) presented an Open Grid Services Architecture that addresses the challenges to achieve various qualities of service when running applications on top of different native platforms. The architecture builds on concepts and technologies from the Grid and Web services communities. The architecture defines a uniform exposed service semantics; defines standard mechanisms for creating, naming, and discovering transient Grid service instances; provides location transparency and multiple protocol bindings for service instances; and supports integration with underlying native platform facilities. The Open Grid Services Architecture also defines interfaces and associated conventions, mechanisms required for creating and composing sophisticated distributed systems, including lifetime management, change management, and notification. Service bindings can support reliable invocation, authentication, authorization, and delegation, if required. The resource sharing mechanisms are complementary to the Grid services, but can be incorporated to implement a service-oriented architecture.

3 AgentScape: A Scalable Multi-Agent Infrastructure

AgentScape is a middleware layer that supports large-scale agent systems. The rationale behind the design decisions are (i) to provide a platform for large-scale agent systems, (ii) support multiple code bases and operating systems, and (iii) interoperability with other agent platforms. The consequences of the design rationale with respect to agents and objects, interaction, mobility, security and authorization, and services are presented in the following subsections.

3.1 The AgentScape Model

The overall design philosophy is "less is more," that is, the AgentScape middleware should provide a minimal but sufficient support for agent applications, and "one size does not fit all," that is, the middleware should be adaptive or reconfigurable such that it can be tailored to a specific application (class) or operating system/hardware platform.

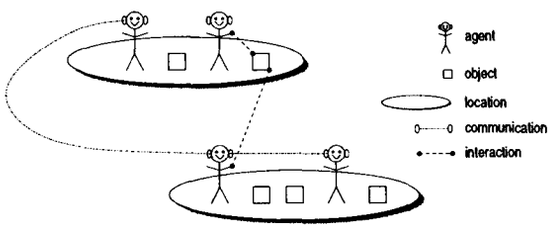


Figure 1: AgentScape conceptual model.

Agents and *objects* are basic entities in AgentScape. A *location* is a "place" in which agents and objects can reside (see Fig. 1). Agents are active entities in AgentScape that interact with each other by message-passing communication. Furthermore, agent migration in the form of weak mobility is supported (Picco, 2001). Objects are passive entities that are only engaged into computations reactively on an agent's initiative. Besides agents, objects, and locations, the AgentScape model also defines *services*. Services provide information or activities on behalf of agents or the AgentScape middleware.

Scalability, heterogeneity, and interoperability are important principles underlying the design of AgentScape. The design of AgentScape includes the design of agents, objects and services, interactions, migrations, security and authorization, as well as the agent platform itself. For example, scalability of agents and objects is realized by distributing objects according to a per-object distribution strategy, but not agents. Instead, agents have a public representation that may be distributed if necessary.

The basic idea in the AgentScape model is that most of the functionality is provided by the *agent interface* implementations such that the middleware (or the agent representation of the middleware) can be designed to perform basic functions. This approach has a number of advantages. First as the middleware must provide basic functionality, the complexity of the design of the middleware can be kept manageable and qualities like robustness and security of the middleware can be more easily asserted. Additional functionality can be implemented in the agent-specific interface implementation (see Fig. 2).

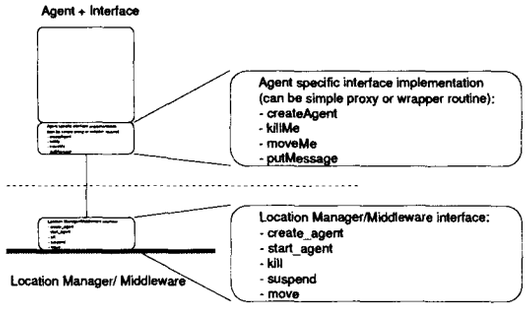


Figure 2: The AgentScape interface model.

Agent-agent interaction is exclusively via message-passing communication. Asynchronous message passing has good scalability characteristics with a minimum of synchronization between the agents. Tuple spaces also provide a mechanism for communication that does not enforce synchronization between the communicating partners, but also cannot enforce the actual receipt of the information.

Agent migration between locations is based on weak mobility. The *state* of the agent is captured (e.g., the variables referenced by the agent) but not the *context* of the agent (e.g., stack pointer and program counter).

3.2 An AgentScape Architecture

The four basic concepts agents, objects, locations, and services are further implemented in the AgentScape architecture. Agents and objects are supported by *agent servers* and *object servers* respectively. Agent servers provide the interface and access to AgentScape to the agents that are hosted by the agent server. Similarly, objects servers provide access to the objects that are hosted by the object server. A location is a closely related collection of agent and object servers, possibly on the same (high-speed) network, on hosts which are managed in the same administrative domain.

Depending on the policy or resource requirements, one agent can be exclusively assigned to one agent server, or a pool of agents can be hosted by one agent server. The explicit use of agent servers makes some aspects in the life cycle model of agents more clear. An active agent is assigned to, and runs on a server; a suspended agent is not assigned to an agent server. In this model, starting a newly created, or activating an existing suspended agent, is similar, and some design decisions of the agent life cycle can be simplified.

The use of agent and object servers is transparent to the agents. Hence, agent servers do not belong to the AgentScape model from the agent perspective. However, an agent could ask the middleware to determine on which agent server the agent runs.

The *AgentScape Operating System* (AOS) forms the basic fundament of the AgentScape middleware. An overview of the AgentScape architecture is shown in Fig. 3. The AOS offers a uniform and transparent interface to the underlying resources and hides various aspects of network environments, communication, operating system, access to resources, etc. The AgentScape API is the interface to the middleware. Both agents and services (e.g., resource management and directory services) use this interface to the AOS middleware.

The design of the AgentScape Operating System is *modular*. The AOS kernel is the central active entity that coordinate all activities in the middleware. The modules in the AOS middleware provide the basic functionality. Below a brief overview of the most important modules is given. The life-cycle module is responsible for the creation and deletion of agents. The communication module implements a number of communication services, e.g., similar to UDP, TCP, and streaming, with different qualities-of-service. Support for agent mobility is implemented in the migration module. The location service associates an agent identifier with an address (or contact-point). There are also location services for objects and locations. The security architecture is essential in the design of AgentScape, as it is an integral part of the middleware. Many components in the middleware have to request authentication or authorization in order to execute

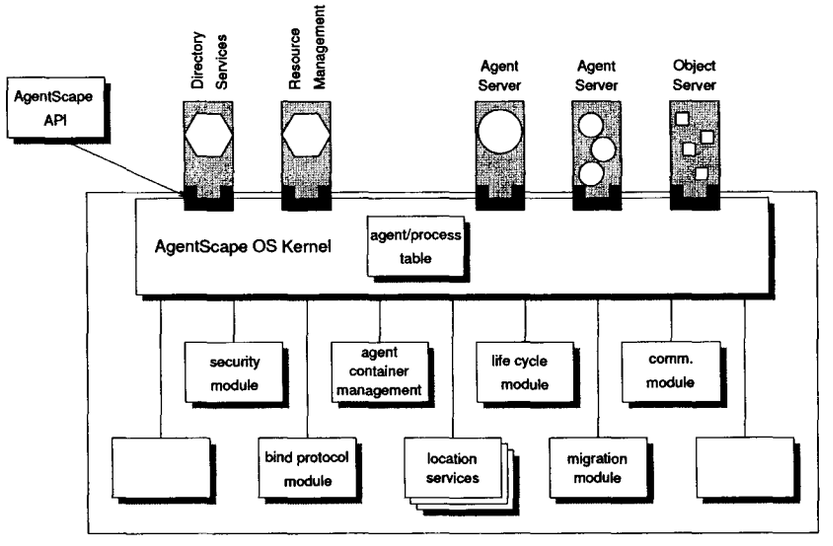


Figure 3: An AgentScape middleware architecture.

their tasks.

In AgentScape, interoperability between agent platforms can be realized in two ways. First by conforming to standards like FIPA or OMG MASIF. These agent platform standards define interfaces and protocols for interoperability between different agent platform implementations. For example, the OMG MASIF standard defines agent management, agent tracking (naming services), and agent transport amongst others. The FIPA standard is more comprehensive in that it defines also agent communication and agent message transport, and even defines an abstract architecture of the agent platform. A second approach to interoperability is realized by reconfiguration or adaptation of the mobile agent. This can be accomplished by an agent factory as described by Brazier et al. (2002), which regenerates an agent given a blueprint of the agent's functionality and its state, using the appropriate components for interoperability with the other agent platform.

3.3 AgentScape Prototype

The current prototype implementation of the AgentScape architecture provides the following basic functionality: creation and deletion of agents, communication between agents and middleware, and weak migration of agents. The AgentScape Operating System kernel and some basic services are implemented in the programming language Python, while the agent server is implemented in Java. As a proof of concept, the middleware not only supports agents written in different programming languages, but its components are implemented in different programming languages.

4 Supporting Agent-Based Grid Management

Resource management is one of the central components of wide-area distributed computing systems like a grid architecture. There have been various projects focused on grid computing that have designed and implemented resource management systems with a variety of architectures and services. Krauter et al. (2002) describe a comprehensive taxonomy for resource management architectures. The design objectives and target applications for a Grid motivate the architecture of the resource management system.

Decentralized, peer-to-peer interaction, resource trading, and machine learning are typically application areas where multi-agent systems are a potentially effective solution. Based on the definition of, for example, the Open Grid Services Architecture (Foster et al., 2002), multi-agent systems can be integrated with grid environments to provide services such as resource management. To integrate multi-agent systems, middleware support for agents should also be integrated with the grid environment.

For the interoperability of AgentScape and a grid environment, or more specifically a grid middleware, two levels of interoperability are important: *runtime system* and *middleware level.*

First, interoperability, extensibility and adaptivity at the runtime system level is incorporated in the agent interface, that is, code associated with an agent's implementation. Agents can provide their own implementation of a runtime system which makes calls on the middleware. This makes the agents adaptable to different environments and extensible if other runtime services are required. The interfaces can be loaded dynamically, and after migration

of the agent to another platform, a platform specific interface implementation can be bound to the agent. This flexibility makes the agent highly adaptive and extendible.

Second, interoperability and adaptivity at the middleware level is provided by the component-based design of the incorporated functionality of the middleware. That is, the components that implement the required functionality have a clearly defined interface, and can be replaced with other implementations. For example, the standard communication module that is included with AgentScape, can be replaced by a communication module supported by the grid middleware. This is also according the Globus design philosophy where existing or proprietary technologies can be incorporated in the Globus system (Foster et al., 2001).

Agent-based methods for coordination and control mechanisms in heterogeneous distributed system is a new and active research area. Minsky and Ungureanu (2000) formulate the requirements for agent-based coordination and control: (i) coordination policies need to be enforced; (ii) the enforcement needs to be decentralized; (iii) coordination policies need to be formulated explicitly; and (iv) it should be possible to deploy and enforce a policy incrementally. Minsky and Ungureanu describe a law-governed interaction mechanism that satisfy these principles, and can be used as a model for an agent-based approach to coordination and control of resource management system.

5 Summary and Future Work

Computational grids are often used for computationally intensive applications. Grids are currently expanding to Internet-scale sizes. Management issues, including task allocation and resource management, becomes a very important issue. Agent-based approaches may facilitate the management of these large-scale grids. Unfortunately, almost all of the current agent-based systems are not developed for large-scale environments; CoABS is a notable exception.

AgentScape, a large-scale distributed agent system, designed to support heterogeneity and interoperability, facilitates extensibility: it is relatively easy to build agent environments "on top of" AgentScape. AgentScape is also relatively easily adapted to different (lower-level) operating systems and network infrastructures. As such, AgentScape can be relatively easily integrated with other environments and support agent-based approaches to grid resource management.

Future work is further development of the AgentScape prototype. An elaborate management system will be incorporated to deal with performance, security, fault-tolerance, and accounting. Other research issues are scalable services for agents, such as name, location, and directory services. Agent-based scheduling and resource allocation algorithms have to be developed and evaluated on the AgentScape middleware.

References

J. M. Bradshaw, N. Suri, A. J. Cañas, R. David, K. Ford, R. Hoffman, R. Jeffers, and T. Reichherzer. Terraforming cyberspace. *Computer*, 34(7):48–56, July 2001.

F. M. T. Brazier, B. J. Overeinder, M. van Steen, and N. J. E. Wijngaards. Agent factory: Generative migration of mobile agents in heterogeneous environments. In *Proceedings of the ACM Symposium on Applied Computing (SAC 2002)*, Madrid, Spain, March 2002.

I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke. The physiology of the Grid: An open Grid services architecture for distributed systems integration. `http://www.globus.org/research/papers/-ogsa.pdf`, January 2002.

I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the Grid: Enabling scalable virtual organizations. *International Journal on High Performance Computing Applications*, 15 (3):200–222, Fall 2001.

R. Ghanea-Hercock, J. C. Collis, and D. T. Ndumu. Cooperating mobile agents for distributed parallel processings. In *Proceedings of the Third Annual Conference on Autonomous Agents*, pages 398–399, Seatle, WA, April 1999.

K. Krauter, R. Buyya, and M. Maheswaran. A taxonomy and survey of grid resource management systems for distributed computing. *Software: Practice and Experience*, 32(2):135–164, February 2002.

F. Manola and C. Thompson. Characterizing the agent grid. `http://www.objs.com/agility/tech-reports/990623-characterizing-the-agent-grid.html`, June 1999.

N. Minsky and V. Ungureanu. Law-governed interaction: A coordination and control mechanism for heterogeneous distributed systems. *ACM Transactions on Software Engineering and Methodology*, 9(3):273–305, July 2000.

G. P. Picco. Mobile agents: An introduction. *Microprocessors and Microsystems*, 25(2):65–74, April 2001.

O. F. Rana and D. W. Walker. 'The Agent Grid': Agent-based resource integration in PSEs. In *Proceedings of the 16th IMACS World Congress on Scientific Computing, Applied Mathematics and Simulation*, Lausanne, Switzerland, August 2000.

N. J. E. Wijngaards, B. J. Overeinder, M. van Steen, and F. M. T. Brazier. Supporting Internet-scale multi-agent systems. *Data and Knowledge Engineering*, 2002. (To appear).

M. J. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2): 115–152, 1995.

RAGE: an agent framework for easy distributed computing

P. Mathieu, J.C. Routier, and Y. Secq

Laboratoire d'Informatique Fondamentale de Lille - CNRS UPRESA 8022
UNIVERSITÉ DES SCIENCES ET TECHNOLOGIES DE LILLE
59657 Villeneuve d'Ascq Cedex
{mathieu,routier,secq}@lifl.fr

Abstract. This paper presents the RAGE framework. RAGE stands for *Reckoner AGEnts*. It is an agent framework for the *easy* design of distributed computing environment. RAGE has been developped using the MAGIQUE multi-agent framework. Because distribution, cooperation and organization are key concepts in multi-agent systems, they are natural solutions for the design of such applications. The use of MAGIQUE has contributed to an easy development of RAGE and provides to the application the capacity to smoothly evolve and adapt. The goal of this article is not only to present RAGE but also to show how the development and evolutions of such an application with a multi-agent system is simple.

1 Introduction

One application field of multi-agent systems (MAS) is *Distributed Problem Solving*, and one typical example of such problems is the distributed calculus. Because distribution, cooperation and organizations are key concepts in MAS, they are natural solutions for the design of applications of distributed computing.

The use of a multi-agent framework allows the designer to get rid of the problems that would have been generated by the distribution and the communications between (what would have been) clients. Indeed these are primitives in multi-agent infrastructures and are often even hidden to the designer. Moreover, the agent approach (or paradigm) leads the designer to a natural decomposition of the problem in term of *tasks* and *roles*. The notion of *role* has been largely emphathized in works related to actor languages [7], subject oriented programming [6], and of course in the multi-agent field [4]. This notion relies on the specification of the behaviour of an actor/agent. In a sense, roles can be seen as an equivalent of interfaces (or pure abstract classes) in object oriented programming. The main difference resides in the fact that interactions are not constrained with interfaces (anybody can invoke a method), while with *roles* one can ensure that the request comes from the right *role*. The other point with *roles* is that they identify the functional requirements and are not tied to particular agents.

For all these reasons we claim that multi-agent systems are appropriate infrastructures for the design and development of flexible framework for distributed calculus applications.

This paradigm of dynamic construction of agent from skills, has several advantages :

- *development becomes easier* : modularity is given by the skills,
- *efficiency* : skills distribution can be dynamilly adapted,
- *robustness* : critical skill can be preserved,
- *autonomy and evolutivity* : runtime customization available through adaptation to runtime environment.

The organisational model. In MAGIQUE, there exists a basic default organisational structure which is a hierarchy. It offers the opportunity to have a default automatic delegation mechanism to find a skill provider.

The hierarchy characterizes the basic structure of acquaintances in the MAS and provides a default support for the routing of messages between agents. A hierarchical link denotes a communication channel between the implied agents. When two agents of a same structure are exchanging a message, by default it goes through the tree structure.

With only hierarchical communication, the organisation would be too rigid, thus MAGIQUE offers the possibility to create direct links (i.e. outside the hierarchy structure) between agents. We call them *acquaintance links* (by opposition of the default *hierarchical links*). The decision to create such links depends on some agent policy. However the intended goal is the following: if some request for a skill occurs frequently between two agents, the agent can take the decision to dynamically create an acquaintance link for that skill. The aim is of course to promote the "natural" interactions between agents at the expense of the hierarchical ones.

With the default acquaintance structure, an automatic mechanism for the delegation of request between agents is provided. When an agent wants to exploit some skill it does not matter if he knows it or not. In both cases the way he invokes the skills is the same. If the realization of a skill must be delegate to another, this is done transparently for him, even if he does not have a peculiar acquaintance for it. The principle of the skill provider search is the following:

- the agent knows the skill, he uses it directly
- if he does not, several cases can happen
  - first he has a particular acquaintance for this skill, this acquaintance is used to achieve the skill (ie. to provide service) for him,
  - he is a supervisor and someone in his hierarchy knows the skill, then he forwards (recursively through the hierarchy) the realisation to the skilled agent,
  - he asks its supervisor to find for him some gifted agent and his supervisor applies the same delegation scheme.

One first advantage of this mechanism of skill achievement delegation is to increase the reliability of the multi-agent system: the particular agent who will perform the skill has no importance for the "caller", therefore he can change

inherits some predefined patterns). The user has mainly to know two concepts: what is a *task* and what is a a *result*. A *task* can be seen as the algorithm that is distributed, while a *result* represents the data produced by the *task* and which must be stored.

The user of the framework has to define what should be distributed, and what is the global scheduling of its main algorithm. The *task* defines the chunk of algorithm that will be distributed among agents. The simplest way for the user to create a *task* is to subclass the `AbstractTask` class and to define the only two methods:

```
abstract public void compute();
abstract public boolean finished();
```

The complexity for the user is then not bigger than writing a JAVA Applet. Therefore, the end user has a rather OO view of the framework: he extends one class to tailor it to his distributed application and uses the underlying framework without necessary explicitly knowing what happens.

3.2 Implementation with Magique: the designer point of view

The design of the framework has loosely followed the GAIA methodology [12] and has been defined through those steps: first, definition of the roles involved in the framework and their abilities (or skills), second, definition of the organization of roles in the system and finally, the mapping of roles on agents.

The first step identifies the main entities of the application, and the abilities they have to assume. This step describes also the interactions between roles. The second step defines the number of agents playing a particular role and their position in the organization. Then, the last step do the mapping between the agents and the role(s) they play.

The first task is to define the roles. A short analysis of the problem allows to determine a set of roles that can be distinguished to bring a solution to the problem of distributed calculus. Here is a short description of these roles:

Boss role is responsible of all the interactions with the user part
Task Dispatcher role has to dispatch *tasks* and deal with the fault tolerancy.
Platform Manager role manages the agents that will compute the *tasks*, and must ensure that there are always available *tasks* for these agents.
Reckoner Agent role is the worker of the framework, it computes *tasks*, and send back the *results* of this computation.
Repositories Manager role manages the storage of *results*.
Result Repository role is a mirror of the database of *results*.

As we implemented the system with MAGIQUE, we have put in concrete form those roles with the corresponding skills. For example, the *Platform Manager role* is defined by a MAGIQUE skill that implements its goal: getting tasks and dispatching them to *Reckoner Agents*. A role is then defined by a set of skills that

Here we have briefly seen the designer vision: the determination of roles and their interactions and then the choice of the organization. The agent oriented approach has allowed a quick development of the application. Moreover, we see that even if finally the user has not necessarily an agent vision of the application, the design was agent oriented.

3.3 Experiments done with RAGE

To evaluate the framework, we have done some experiments that are ranging from simple examples to complex applications. The first experiment is a *tutorial* example that computes an approximation of PI. It is based on a Monte Carlo method and illustrates how simple it is to create a *task* and to run the framework. The second tutorial experiment is a *naive* implementation for prime number decomposition, which enabled us to evaluate the scaling of the infrastructure. The third sample is bigger, it is an exploration of the underlying structure of the Donkey sliding block game. The algorithm consists in an exhaustive generation of game states graphs[2]. It is interesting as it works in two stages and shows how computation can be canceled. The last sample is an application for solid mechanics : it is an implementation of the two-dimensional displacement discontinuity method[8]. Those experiments along with the framework can be downloaded at `http://www.lifl.fr/SMAC/projects/magique/examples`.

4 Conclusion

The presented application, RAGE, is an easy to use framework for distributed computing. It allows to develop, distribute and run calculus on heterogeneous framework with no need of background in distributed computing or agents technologies. The user can only direct its efforts towards his very calculus.

This article argues that multi-agent paradigms: agent animacy, organization and roles, are key notions to support the analysis, design and implementation of open large-scale distributed systems. It is particulary significant that those application models are in adequation with multi-agent paradigms. It has been illustrated with an application of distributed computing, but could be extended to other classes of applications (like CSCW[10]).

From our point of view, RAGE demonstrates that agent oriented programming is an appropriate framework for the development of such distributed applications. And as a consequence, the resulting framework is easy, first, for the user who has a calculus to do, and, second, for the designer who wants to extend the capacities of the framework.

References

1. N.E. Bensaid and P. Mathieu. A hybrid and hierarchical multi-agent architecture model. In *Proceedings of PAAM'97*, pages 145–155, 1997.

Laying the Foundations for the Semantic Grid

Steven Newhouse, Anthony Mayer, Nathalie Furmento,
Stephen McGough, James Stanton and John Darlington
London e-Science Centre,
Imperial College of Science, Technology and Medicine,
180 Queen's Gate,
London SW7 2BZ, UK
icpc-sw@doc.ic.ac.uk
http://www-icpc.doc.ic.ac.uk/components/

Abstract

Information relating to the resources, applications and the user's wishes are key to the transparent and effective exploitation of the federated resources within Computational Grids. These federated data, computational or software resources are owned by real organisations and made available as services to different computational communities or virtual organisations spanning multiple administrative boundaries. Higher-level services use the information relating to the resources' capability and the mechanisms for service fulfilment to automatically discover interoperable services and select the most appropriate service for the user with minimal human intervention.

Within this paper we describe a service-oriented Grid architecture that utilises existing web service protocols to federate resources into computational communities. The service-oriented information contained in the computational communities is exploited by higher-level services to ensure effective utilisation of the resources by both its providers (the resource owners) and consumers (the users). We believe the systematic definition, presentation and exploitation of such information constitutes the first step towards the construction of a Semantic Grid.

1 Background

Computational Grids have provoked widespread interest within the scientific and engineering community over the last decade (1). There are now many global projects focussing on the development of Grid middleware and the mechanisms needed by the applied science community to effectively exploit these infrastructures (2). These projects are all being driven by the needs of a diverse applied science community involving both data and compute intensive applications (e.g. EU Datagrid, PPDG, Griphyn, NASA's Information Power Grid).

Heterogeneous resources within these Grids are forged into a single virtual organisation through the use of middleware. The Grid middleware collects and publishes information relating to the resources within the organisation while providing secure platform neutral interfaces to the underlying resources. Globus is one such widely deployed middleware that uses the Metacomputing Directory Service (MDS) to hold and distribute information through a hierarchical network of MDS server within a virtual organisation (3). Access to the resources within a virtual organisation is controlled through a 'gatekeeper' which verifies a user's identity through an X.509 public key certificate.

The specification, development and standardisation of Grid related protocols is taking place through the Global Grid Forum (http://www.gridforum.org/), formed through the merging of activities in North America, Europe and Asia. Its role, through the activity of its research and working groups in areas such as security, information management, applications, etc., is to provide a collaborative forum for researchers in industry, computer science and the applied science communities. Its meetings now attract several hundred researchers from around the world.

The activity within the Grid community can be compared to that within the Web community a decade ago. At that time there was an active worldwide research community developing competing and incompatible protocols, and innovative functionality within the server and client browsers. Coordination and standardisation of these activities was left by the community to the World Wide Web Consortium (W3C). Since 1994 it has guided the evolution of the Hyper-text Markup Language (HTML) and produced the several new standards such as the Extensible Markup Language (XML).

The recent emergence of business to business e-commerce has exposed the limited capabilities of existing web protocols when used to develop higher-level services. Currently, the division between page content and its presentation is frequently blurred within HTML encoded web pages. The W3C has been instrumental in developing approaches that enable a clear separation between the content (encoded as an XML schema) and its visual repre-

rescheduling as 'better' resources become available or computational steering of the application (6).

3 Grid Programming Model

It is impossible to proscribe a Grid programming language on the diverse applications within the e-science community. Likewise, it is difficult to proscribe a programming model unless it provides the flexibility to encompass existing legacy applications and those compatible with current software engineering practices.

We are prototyping a component based model that allows legacy applications to be encapsulated as a single component with defined interfaces (7; 8). Our model allows a component's interface to be matched to any number of valid implementations. The decoupling between a components interface and implementation allows the component to be deployed on the 'best' currently available execution platforms within a distributed Grid environment (9). For example, each component may have several implementations such as different algorithms (e.g. iterative or direct solvers) for different platforms (e.g. Solaris or Linux) for different architectures (e.g. serial or parallel). We describe the component interface, its implementations and capabilities through CXML - an XML schema for Component applications (10)

More generally, we define an application as a network of linked components where we are able to describe the frequency and data volume of their interactions. By combining our knowledge of these interactions with information as to how a component's implementation will behave on a particular platform, with data relating to the platforms current state, we are able to optimise its performance within a particular policy (5). The decoupling between an interface and its implementation also allows us, by storing persistent data outside of the component, to migrate an application's components to other resources during execution.

In our model, a user submits a job (as an application specification) to the 'Grid' by placing a CXML description of the application network and the user requirements into the public computational community. The application mapper matches the components used in the application to the implementations that exist within the resources in the computational community. The application network is instantiated on distributed resources with the best component implementations to maximise a user defined criteria. The effective deployment of the application across distributed resources is enabled by the rich metadata relating to the resources, application structure and implementations available to the higher-level services. A component expects to be deployed into a local 'Grid container' that provides access to a minimal set of basic services. The detailed definition of these container services is currently under investigation.

4 Grid Services

Our experiences with ICENI and other infrastructures, such as Globus, have demonstrated the need for a minimum set of services to support e-science within computational Grids. We characterise these services into two groups: low-level services that interface directly to the underlying Grid fabric or provide essential services and higher-level services utilising these lower-level services. All services are registered with a Registry Service in the local computational community that exchanges information with its peers in other organisations.

4.1 Registry Service

The Registry Service (like the Jini Lookup Service) provides a 'known' point for information exchange between clients and other services. The local private registry service within an organisation federates its internal services into a community wide infrastructure through the actions of the domain manager. The domain manager 'pushes' information relating to capability and usage policies of the local community's services to (potentially) several public community wide Registry Services within different virtual organisations.

The domain managers and community wide Registry Service can be viewed as part of a peer-to-peer network. We intend to use this structure to propagate the services within one community to other computational communities. Propagation continues while there is an intersection between the acceptable useage policies of services from the remote domain with the local user community. This approach should ensure that all users allowed to use a resource will have access to it providing there is at least an indirect link between the two communities. Modelling of this infromation structure and an examination of the service propagation distance is underway.

4.2 Low-level Services

The local private Registry Service will contain a number of low-level services:

- Service Register – provides a mechanism for service registration, discovery and instantiation
- Authentication – verification of the user or host using X.509 certificates
- Authorisation – expression of a resources' usage and access control policy
- Execution Resources – invoke a deployed component on a resource
- Software Resources – an index of the locally deployed software components

the Jini service discovery mechanisms. The ICENI services are made available to clients using a web service protocol through a proxy embedded within an application server. See Figure 2. Both the internal ICENI and SOAP encoded web services messages use the same XML schema to describe the message content.

6 The Semantic Grid

The Grid middleware now being developed to support e-Science within the UK and elsewhere is evolving into a service-oriented architecture built upon standard web protocols such as XML, SOAP, WSDL and UDDI. While developing protocols within such a framework promotes interoperability there is no certainty that these protocols will always be understood. It is here, we feel, that a Semantic Grid will start to develop as services become capable of discovery and self-organisation.

Tim Berners-Lee defines the Semantic Web as 'an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation.' (12). Likewise, the Semantic Grid can be described as an 'extension of the current Grid in which information and services are given well-defined meaning, better enabling computers and people to work in cooperation'. In such an environment it is essential that information relating to the needs of the user and their applications, and the resource providers and their networking, storage and computational resources all have easily discovered and defined meaning that can be used by higher-level services to effectively exploit the Grid.

The development of these service-oriented ontologies will enable compatible services to autonomously build the large complex distributed computing environment that constitutes the Grid. Fundamental to this goal is the expression of Grid services through well defined protocols. From these protocols we are able to understand the capabilities of the service and, potentially, reason as to how it can interact with other services to meet the needs of the users. This reasoning can take place at several levels, from service composition and application assembly to the representation and exploitation of knowledge generated through data and computation services (13). The mechanisms for exploiting the service and data ontologies within the Semantic Grid may range from simple schedulers, application mappers and resource brokers to sophisticated, autonomous, mobile and intelligent agents.

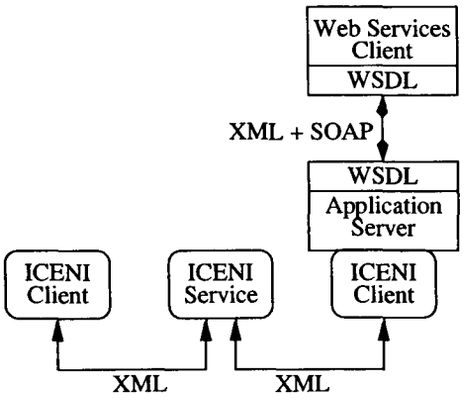


Figure 2: Extending ICENI to support interaction with web services.

7 Conclusions

Computational Grids are demonstrating a convergence of many existing areas of computer science research. The high performance computing community are developing new algorithms to support heterogeneous wide area computation between different supercomputers. Applications are having to deploy to new resources expecting only the services available within a 'standard container' environment. Users and applications are discovering distributed resources and services maintained through a network of peer-to-peer directory services. The autonomous discovery and assembly of a Grid environment from the available services is reducing the complexity involved in constructing a complex software environment while improving robustness through multiple services.

We consider the movement towards a Semantic Grid (at both the service and knowledge layers) as essential in simplifying the effective utilisation of sophisticated distributed services. The description of these services (using existing web-service protocols) will enable their intelligent composition and exploitation with minimal human interaction. The transparent and optimal delivery of sophisticated computational and data services to the applied science community will be key to the successful adoption of e-science.

We have shown how ICENI will continue to use Jini as a registration and service discovery mechanism while exposing its functionality within a web services framework to promote interoperability with other infrastructures. The XML schemas governing the interaction between different ICENI services will continue to be developed and encapsulated within SOAP for use by the web services interface. Services within our computational communities will be propagated to other organisations through a peer-to-peer mechanism.

Acknowledgements

Steven Newhouse gratefully acknowledge the formal and informal discussions that have taken place within the Architectural Task Force of the UK e-Science Core Programme which have helped clarify some if the issues discussed within this paper.