Self Managing Middleware for Dynamic Grids

Sachin Wasnik, Terence Harmer, Paul Donachy, Andrew Carson, Peter Wright, John Hawkins, Christina Cunningham, and Ron Perrott

Belfast e-Science Centre, The Queen's University of Belfast,
Belfast, BT7 1NN, UK
{s.wasnik, t.harmer, p.donachy, a.carson, pwright04, j.hawkins, christina.cunningham, r.perrott}@qub.ac.uk

Abstract. As grid infrastructures become more dynamic in order to cope with the uncertainty of demand, they are becoming extremely difficult to manage. At the Belfast e-Science Centre, we are attempting to address this issue by developing Self Managing Grid Middleware. This paper gives an overview of the middleware and focuses on the design, implementation and evaluation of a Resource Manager. Also in this paper we will see how our approach, which is based on federated UDDI registries, has enabled us to implement some of the desired features of next generation grid software.

Keywords: Grid Computing, UDDI Registries, Grid Resource Manager, SLA.

1 Introduction

Most production Grids [1], irrespective of whether they are being deployed in commercial or academic environments, must cope with variation in demand. A goal for next generation Grid research and development is to produce a "...fully distributed, dynamically reconfigurable, scalable and autonomous infrastructure to provide location independent, pervasive, reliable, secure and efficient access to a coordinated set of services encapsulating and virtualizing resources (computing power, store, instruments, data etc) in order to generate knowledge", according to the CoreGrid European Network of Excellence [2]. There has been a significant improvement in focus of the vision of Grid Computing [3] since the term was introduced. A vital improvement still to be implemented satisfactorily is to make Grid Computing more dynamic so that it is able to cope with uncertainty of demand. Some recent work including HAND [4] and Dynamic Deployments [5] has focused on dynamically deploying and scaling Grids in production as and when needed.

The term "autonomic computing" is representative of a vast and somewhat tangled hierarchy of natural self governing systems, which consist of many interacting, self governing components that are often compromised of a large number of interacting, autonomous self governing components at the next level down. According to the vision of Autonomic Computing [6], the self-managing systems feature automatic mechanisms for operator free maintenance of stand alone and distributed resources, including self-configuration, self optimization, self-healing, self-protection and others. This vision overlaps in its' goals with the pursuits of adaptability and dependability as described above in the recent definition of Grid Computing.

In particular, the adaptability of Grids can be interpreted as self-management on a different scale (and environment), thus making it worthwhile to exploit the discovered approaches in both domains. On the other hand, dependability mechanisms share a lot of scenario problems and approaches with self-management mechanisms (e.g. automatic fault recovery and preventive management actions such as software rejuvenation), thus calling for a convergence of research in these areas.

Trends in automating Service Level Agreement (SLA) management [7], from SLA creation to the performance monitoring of SLA's, can help the Resource Manager to sense the exact needs of users. With the help of an SLA Manager, middleware can act as a biological system which can sense and respond to the needs of the user. This should enable the effective utilization of resources by dynamically deploying, un-deploying and reconfiguring resources as and when needed. In such an infrastructure, Resource Managers are not only responsible for managing the resources, but also for selecting the resources on which the applications are to be deployed on. Thus the Resource Manager can act as the backbone of the self managing grid middleware.

Although a centralized Resource Manager can be very useful for a small number of resources, it may not be able to scale as the number of resources increases. A centralized Resource Manager acts as single point of failure and is vulnerable to security attacks. A decentralized Resource Manager can provide fault tolerance for the middleware by devolving responsibilities to a number of Resource Managers interacting with each other. A decentralized Resource Manager provides us with the necessary backbone of the next generation grid middleware but it is also difficult to maintain. This is where the self managing approach can assist in enabling the development of middleware which is self configuring, self healing, self optimizing and self protect.

The rest of the paper is organized as follows. Section 2 describes the architecture of the Self Managing Middleware. Section 3 describes the design and implementation of the federated Registries. Section 4 describes a use case for the middleware followed by the conclusion in section 5.

2 Self Managed Grid Middleware

According to our view of an infrastructure, infrastructure components are organised or grouped into *domains*. The name "domain" attempts to indicate that it is an area of responsibility and also serves to separate this infrastructural component view from other users and organizations ideas such as virtual organizations—a virtual organization might, for example, be built upon a collection of domains as shown in Figure 1.

A domain is a group of computing resources that it is natural to manage collectively; for example, it could be all of the resources in a small organization or it could be the resources in a particular computing rack that share a network

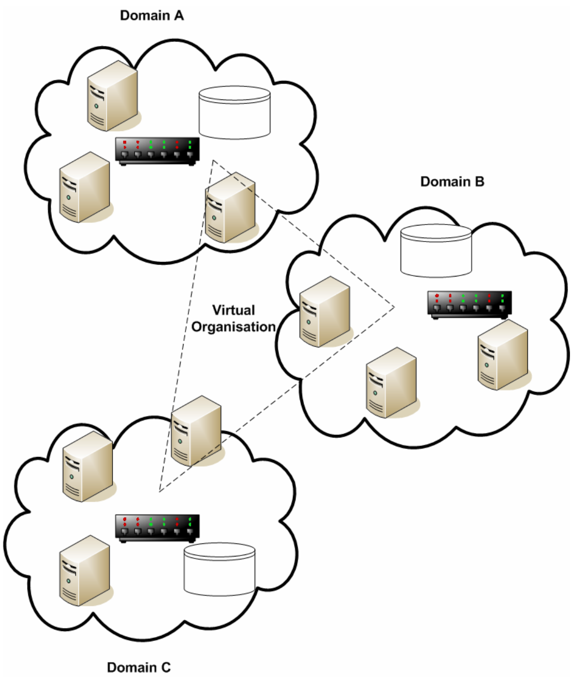

Fig. 1. Different organizations A, B and C forming a virtual organization

connection via a shared network connection or switch. The identification and selection of domains is performed as part of infrastructure design with the intention of identifying natural organizational units. A domain is our mechanism for providing a simple and distributed collection of managed infrastructure components.

The (self) management of grid resources is performed at the domain level. A domain provides a mechanism by which a group of related resources (i.e. services or applications) can be deployed and managed.

A domain may have sub domains. This hierarchical view enables requests to be directed to high-level management components and split between the organization units that are available within a domain—these high level components may enforce local management rules or act as brokers by selecting the best available local domains for deployment.

As shown in figure 2, each domain is managed using the core components of a Software Manager, a Security Manager, a Software Repository and a Resource Manager. A Resource Manger at the domain level is based on a single Registry but at the Grid level, Resource Manager is based on Registry Federation. Resource Manager at the grid level appears as a single logical Resource Manager of all the domains, to which a software manager can issue a single request against multiple Resource Manager and get a single response that contains results based on all the data contained in all the registries.

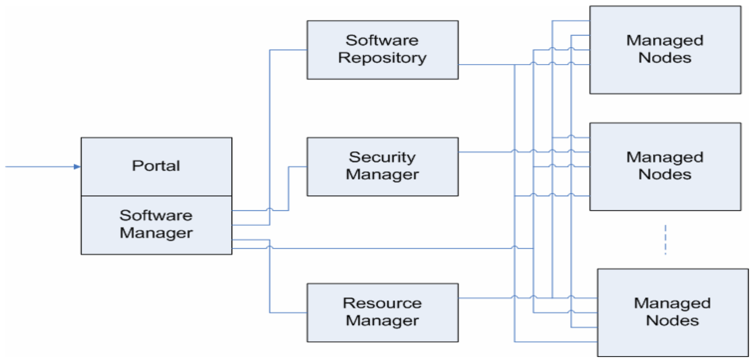

Fig. 2. Managed nodes being directed by the Managers

2.1 Software Manager

The Software Manager component takes a deployment request and performs the specified deployment. A deployment request consists of the deployment action and a configuration definition that enables management of the deployment action. A deployment action can be the installation of software, the execution of a particular application, the deployment of a web/grid service, the un-deployment of an application or web/grid service, the storage of a data source such as a database, the un-deployment of a data source, the recovery of the data held in a data source, or the deployment of a security definition, for example the modification of firewall rules.

The Software Manager may require several deployment actions to fulfil a particular user deployment action; for example, the deployment of a web service may require the deployment of a specific Java environment, a web service container application, applications or web services to support the user web service.

A portal provides a user interface where a user can upload a package by supplying its configuration file—a web service provides the same functionality for an application.

A deployment request may be in one of the following formats:

- A war file
- An RPM
- A resource bundle for Globus container
- A resource bundle for OMII container
- A security configuration schema instance
- A data source bundle
- A meta tar file containing a combination of the above resources

The configuration definition specifies the required environment for the deployment. The action of the Software Manager is to select a suitable host, deploy software to that host that is required, deploy a security and the resources deployed.

An example configuration file for deploying a simple web service might look like this:

```
<config>
 <bundle>
  <summary>
   <bundleType>rpm</bundleType>
   <systemPackageInfo>
    <vendor>none</vendor>
    <name>gridftp_transfer</name>
    <version>2.1</version>
    <description>GridFTP</description>
   </systemPackageInfo>
   <validFrom>12/02/07</validFrom>
   <validTo>12/03/08</validTo>
  </summary>
  <firewall/>
  <callback><url/></callback>
…
```

```
…
  <dependencies>
   <hardware>
    <cpu>
     <speed>1500</speed>
    </cpu>
    <memory>512</memory>
    <storage>
     <freeSpace>15</freeSpace>
     <raid>5</raid>
    </storage>
   </hardware>
   <software/>
  </dependencies>
 </bundle>
</config>
```

2.2 Security Manager

The Security Manager is responsible for configuring and maintaining security on infrastructure components—currently this involves the deployment of digital certificates to enable user and host authentication, updating certificate revocation lists and defining firewall rules.

The Security Manager keeps track of the status of the firewall on each of the managed nodes with the help of an agent installed on them. When a service or application being deployed has a particular security requirement, the Software Manager sends a request to the infrastructure component of the Security Manager which performs the necessary security modifications. A security modification that conflicts with the basic security rules defined for an infrastructure component will cause a deployment request to be rejected; a modification that conflicts with rules deployed to support other applications will result in a different infrastructure component being selected as the deployment target. When a service or application is un-deployed, the security modifications are also un-deployed.

2.3 Software Repository

A Software Repository is maintained to hold different versions of applications and services that can be specified as software dependencies in the configuration file as shown above. When a user submits a configuration file for deployment to the software manager, the Software Repository provides the software to carry out the deployment action. For example, a deployment of war file needs java and a web service container. In this case war file will be provided by the user and the software repository will provide dependent packages of java and web service container.

3 Resource Manager

The convergence of grid computing and service oriented computing has enabled the service registries to take on the role of a Resource Manager [8]. Job scheduling in grid environments has taken a new form relating to the interaction between the service provider and the service consumer, which is shown here in Figure 3.

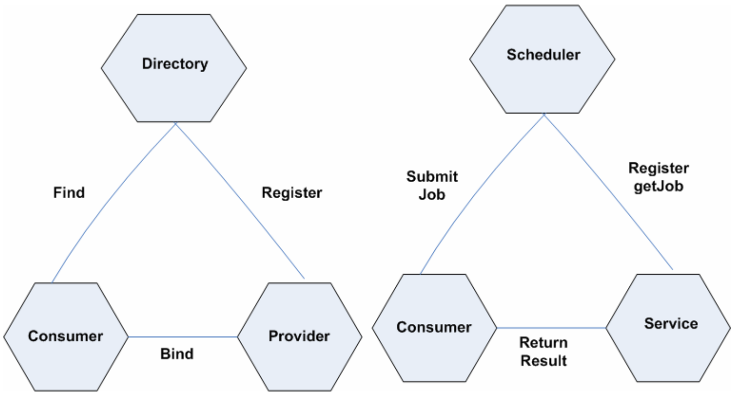

Fig. 3. Interaction diagram showing the interactions between the Service Provider and the Service Consumer

As the user demand on Grids becomes more agile and dynamic, service discovery using static information is not enough and a need emerges for storing Quality of Service (QoS) information inside service registries as well as a complete abstract mapping of compute resources. The compute resources should be mapped in such a way so as to allow a consistent view and management of the resources and this mapping may vary across different infrastructures.

3.1 Resource Mapping

The GLUE Schema [9] is an abstract modelling for Grid resources and mapping to concrete schemas that is being used by most of the production Grids. Glue Schema is widely used in most of the production grids. It has been integrated in number of Grid middleware such as EGEE [10], LCG [11], OSG [12], Globus [13] and NorduGrid [14]. We have represented the GLUE Schema as shown in Figure 4, inside the service registry. A number of specifications for service registries such as UDDI [15], ebXml [16] are available and their implementations are being used for web/grid service discovery. For our middleware we chose the Universal Description, Discovery and Integration (UDDI) registry.

A web/grid ***service*** is represented inside the UDDI registry as a Business Service. A service runs within a compute resource. These compute resources are mapped as Business Entity inside the UDDI registry in a similar way as if they own the service.

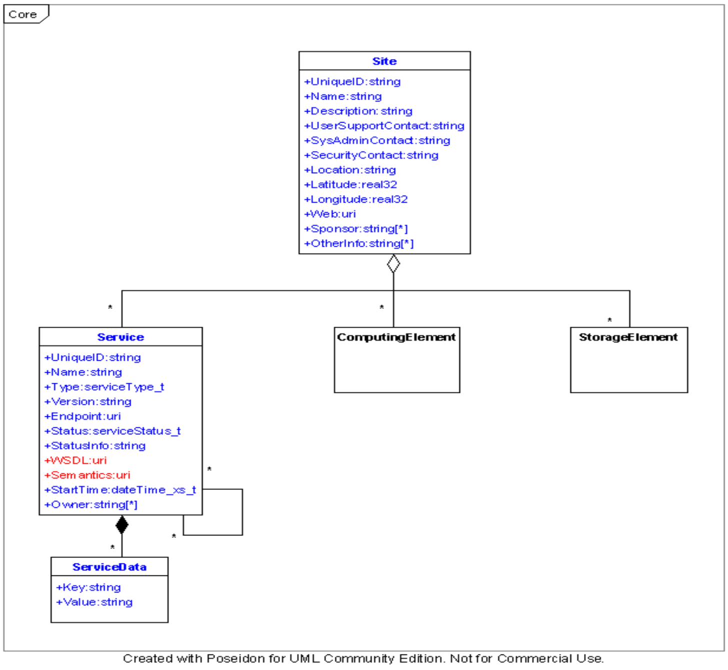

Fig. 4. GLUE Schema

As described in section 2, a site which consists of compute and storage resource is considered as domain which is represented as a business entity. This site business entity can have one or more compute resources and storage resources. The relationship between the machine business entity and the service container business entity is represented as a parent-child relation by using publisher assertions.

3.2 Architecture

An analysis of the individual and collective state of the compute resources can determine the performance of a Grid and enable (self) management activities to respond in an efficient and directed manner; for example, if the Grid is performing poorly then the Resource Manager should identify the compute resources which are contributing to the poor performance and enable the activation of a reactive procedure. The Resource Manager is named as Open Grid Manager (OGM). To achieve the above objective, the OGM for each domain is composed number of components, namely

1) GridManagerAgents (GMA)
2) GridManagerServer (GMS)
3) Web based User Interface (GMUI)
4) UDDI Registry

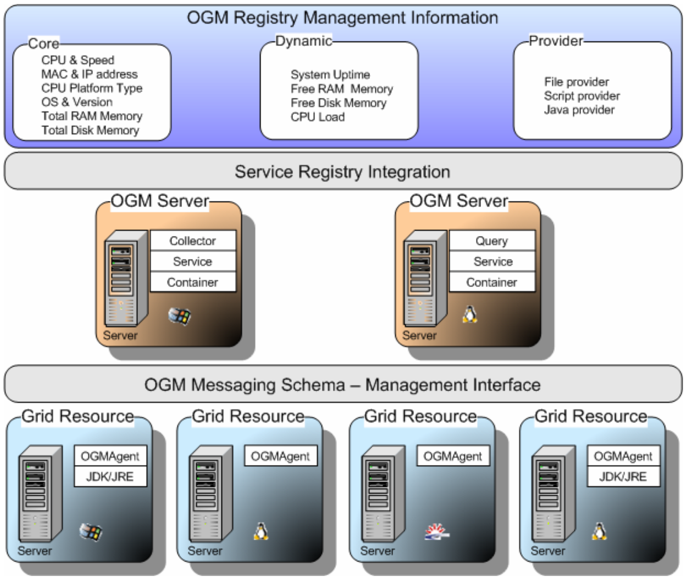

Fig. 5. Architecture of Open Grid Manager (OGM)

The GridManagerServer consists of two services – a Collector Service and a Query Service. The GridManagerAgents are responsible for deducing a machine's state and reporting this to a Collector Service. The Collector Service collects state data from nodes in a distributed environment and forwards this to the UDDI registry.

Each Managed Node registers itself by sending core information to the Collector Service with the help of installed GridManagerAgent. The process of registration is carried out by following steps as shown in the Interaction diagram Figure 6.

1) GridManagerAgent sends the core information to the Collector Service.
2) Collector Service of the GridManagerServer, upon receiving the core information address, makes a create Business Entity call to the UDDI registry.
3) UDDI registry creates a Business Entity and sends back the business key to the Collector Service.
4) Collector Service sends the Business Key back to the GridManagerAgent.

The GridManagerAgent uses the same Business key to continuously update the Business Entity with dynamic information and Provider information. The process of update follows the same steps. The frequency of update is configured via the GridManagerAgent's configuration file.

Apart from the resource information, a collector service also stores information about deployments and un-deployments sent by the Software Manager which is considered as static data, as it doesn't change frequently. Whenever a deployment request is made to the Software Manager, the manager sends the information about the deployment request to the Collector Service. Upon receiving the deployment request and the IP address of the machine on which it is to be deployed, the Collector Service creates a business entity with the resource name, which is a child of the Business Entity representing the machine on which it is deployed.

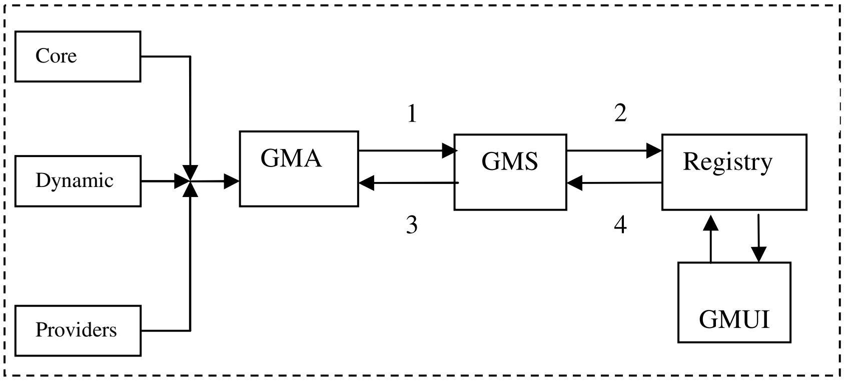

Fig. 6. Interaction between the different components of OGM

For example, while deploying packages such as Grid-FTP, the Software Manager sends information such as the port on which the deployed packages will be running, a username and their associated credentials. When the Collector Service receives this information, it is stored inside the UDDI registry as a Business Entity. These Business Entities have descriptions of transport packages and are children of the Machine Entity on which they are installed.

The Query Service is responsible for answering the queries sent by the software manager. The Software Manager can send queries:

1) To check which machines satisfy certain hardware requirements.
2) To ascertain what packages are already deployed on a given machine. This can help the software manager to discover which machines satisfy the software dependency requirements of a given package to be installed.

To make the domain fault tolerant, the domain operator can keep a backup of their domain registries using database mirroring. In case of a failure of the Resource Manager in a particular domain, a Collector Service and a Query Service is installed and configured to use the stored backup data. Thus the domain manager can roll back to its state just before the failure.

3.3 Federation of Registry

In large distributed grid environments, a single registry can degrade the performance of the whole system as number of clients becomes too large. Also, it becomes a single point of failure, as the whole system depends on the single registry. To make the system more scalable, multiple registries should be utilized.

The latest UDDI version 3 [17] specifications promotes a replication model of data for multiple registries to enable a single view of multiple registries; such a replication model is not suited to the grid environment.

It is preferable that each domain in the federation would have complete autonomy of the data related to the domain. Each domain operator should be able to configure what data to share and with whom it is to be shared. Thus replication between registries owned by multiple independent operators is more complex but more relevant in a

Grid environment which is targeted at cooperating yet independent stake holders. Such a setup requires communication between individual registries to synchronise registration data.

Replication adds communication traffic between the registries for keeping registration data in sync. There is a trade-off between the amount of traffic and the timeliness of the replicated data. If changes to the registration data are propagated to all registries immediately, all registries will have a more or less consistent and current view of the service setup, resulting in a large amount of traffic between the registries. If the registration updates are propagated less frequently and in batches the traffic size decreases (as communication set-up costs are averaged over all changes), but registries will be out of sync for some time. Depending on the application domain, inconsistencies may or may not be tolerable.

Although replication enables scalability, the load is not distributed automatically. Registration is performed at the domain registry but queries can arrive at any of the participating registries. Which registry is to be used is decided by the Query Peer. Load distribution is taken care by the cluster of Query Peers, each of which maintains a list of possible registries. After initial setup, the list could be maintained by automatically updating it with the information from the registry to use.

Each replicated registry keeps a copy of the complete registration data of the whole system. The advantage is that every registry can answer a query by just looking at its database. However a disadvantage of this approach is the large amount of data which may be kept at every site. In our approach, each registry keeps only a subset of the registration data and can only answer query relating to that subset. The data distribution is based on locality.

As the registration data is distributed across registries, multiple registries are involved in answering a query. Orchestrating the devolved registries is performed by the Query Peer which knows all the registries that have answers to their query.

4 Use Case

As part of its core business, a Financial Company analyses Stock Market data from each of the world's main Stock Exchanges. This depends heavily on process and data intensive computations for Risk Management purposes. Feeds are received from each of the exchanges which are fed into a high performance financial database. A number of databases are also maintained containing historical financial data. A number of financial calculations are performed, such as Implied Volatility calculations, on each portfolio managed by the company using the data held in each of the databases.

This system works well for the company on a day-to-day basis. However, to allow them to react more quickly to changes in stock prices as a result of unforeseen major world events, the company would like the option to bring in additional computation power and resources as required. This would enable the Financial Company to react more quickly than their competitors, performing all the additional calculations required to obtain results in near real time, thus gaining a market advantage for their clients.

A system such as the one provided by the Self Managing Middleware described in this paper would clearly benefit this company when they need to react quickly to unpredictable events. Once the increased activity within the stock exchanges has been identified, the company could increase their computational power by quickly deploying additional services to a 3rd party hardware provider and running some calculations from there. This would require transport services to be deployed both at the company's home location and the 3rd party hardware provider's location so that the high performance financial databases could be deployed onto the additional machines. Three databases are required to perform the calculations. One database is required for capturing the data from the live feeds, one database for the intra-day data and another database where historical data is stored. Services which undertake the calculations could then be deployed and initiated, the various calculations performed and the results transported back to the Financial Company for dissemination or use by another application. When the additional capacity was no longer required, the services and databases deployed to the 3rd party hardware provider would be un-deployed.

The Financial Company would be able to impose certain conditions on where their data and services were deployed to. Certain financial regulations imposed upon the Financial Company dictate that the data cannot leave the United Kingdom. The Financial Company may also impose certain restrictions such as 'Don't deploy services or data onto machines owned or managed by one of our competitors'. Information such as this can be included in the configuration file sent with the bundle to be deployed. The Self Managing Middleware enables the Company to have immediate access to additional computational power when required without having to maintain this hardware on a day to day basis.

Secure on-demand provisioning of Risk Management capabilities represents a real and valuable next step for the financial services industry to increase competitiveness and reduce costs. It is also relevant to service provision and consultancy companies currently competing in the international market.

5 Conclusion

In this paper we have discussed the use of Self Managing Software and a Resource Manager to enable the management and control of large-scale grid infrastructures. In the Belfast e-Science centre we have deployed this software in the field for approximately a year and it is an integral part of the testing development of grid of our large-scale commercial projects.

References

1. Foster, I., Gieraltowski, J., Gose, S., et al, : The Grid2003 Production Grid: Principles and Practice. Proc. 13th IEEE Intl. Symposium on High Performance Distributed Computing (2004) 236–245.
2. http://www.coregrid.net
3. Foster, I., Kesselman, C., Nick, J., Tuecke, S., : The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Open Grid Service Infrastructure WG, Global Grid Forum (2002).
4. Qi, Li., Foster, I., Gawor, J.,: HAND: Highly Available Dynamic Deployment Infrastructure for Globus Toolkit 4. Submitted for publication (2006)

5. Watson, P., Fowler, C., Kubicek, C., et al, : Dynamically Deploying Web Services on a Grid using Dynasoar. Proc. 13th IEEE Intl. Symposium on Object And Component-Oriented Real-Time Distributed Computing. ISORC 2006, April (2006)
6. Kephert, J., Chess., D. : The Vision of Autonomic Computing. Computer. Vol. 36 Issue 1. (2003)
7. Sahai, A., Durante, A., Machiraju, V. : Towards automated SLA management for web services. Research Report HPL-2001-310(R.1) Hewlett-Packard laboratories Palo Alto. (2002)
8. Joseph, J., Ernest, M., Fellenstein, C.: Evolution of Grid Computing architecture and Grid adoption models. IBM Syst. J. 43, 624-625 (2004)
9. Andreozzi, S., Burke S., et al: GLUE Schema Specification version 1.2 (2005)
10. Enabling Grids for E-sciencE Project http://www.eu-egee.org/
11. LHC Computing Grid Project http://lcg.web.cern.ch/LCG/
12. Open Science Grid http://www.opensciencegrid.org/
13. http://www.globus.org/
14. http://www.nordugrid.org/
15. Bellwood, T., UDDI Version 2.04 API Specification
16. http://www.ebxml.org
17. Bellwood, T., UDDI Version 3.0 Spec Technical Committee Specification July (2002)