MAGI – Mobile Access to Grid Infrastructure: Bringing the gifts of Grid to Mobile Computing

Ali Sajjad, Hassan Jameel, Umar Kalim, Young-Koo Lee, Sungyoung Lee

Department of Computer Engineering
Kyung Hee University
Yongin-si, Gyeonggi-do
449-701, South Korea
ali@oslab.khu.ac.kr

Abstract: Access to Grid services is currently limited to devices having significant computing, network resources etc. such as desktop computers. On the other hand, most of mobile devices do not have the potential resources to be either direct clients or to host services in the Grid environment. Yet, extending the potential of the Grid to a wider audience promises increase in its flexibility and productivity. In this paper[1] we present the MAGI middleware architecture that addresses the issues of job delegation to a Grid service, adaptive management by including support for offline processing, secure communication, interaction with heterogeneous mobile devices and presentation of results formatted in accordance with the device profiles and limitations. This is achieved by out-sourcing the resource intensive tasks from the mobile device to the middleware. We also demonstrate through formal modeling using Petri Nets that the addi-tion of such a middleware causes minimum overhead and the benefits attained outweigh this overhead.

Keywords: Grid, mobile computing, middleware architecture, distrubuted computing

1 Introduction

Computational entities connected via various kinds of networks can dynamically share their resources and perform computation-extensive tasks by using Grid computing [FKT01]. Extending this potential of the Grid to a wider audience promises increase in flexibility and productivity, particularly for the users of mobile devices who are the prospective consumers and beneficiaries of this technology. Consider a teacher who wants to augment his lecture with a heavy simulation test. He uses his PDA to access a Grid service and submit the request. The service after executing the request compiles the results which are then distributed to the mobile devices of the registered students of that course. Similarly a doctor, on the way to see his patient, requests a Grid medical service to analyze the MRI or CT scans of the patient from his mobile device.

[1] This research has been supported in part by the ITRC program of Korean Ministry of Information and Communication, in collaboration with Sunmoon University, Korea.

By the time he meets his patient; the results would have been compiled and presented on his mobile device to facilitate the treatment. The major bottlenecks affecting mobile devices are battery life, storage, memory, bandwidth, processing power and security. But the clients that interact with the Grid middleware to accomplish a task are required to use client-end, application-specific libraries. These libraries are relatively resource intensive considering the limitations of mobile devices. Conceiving a distributed system that uses these libraries directly will not be a practical system because of the resource demands.

Moreover, most of the conventional distributed applications are developed with the assumption that the end-systems possess sufficient resources for the task at hand and the communication infrastructure is reliable. For the same reason, the middleware technologies for such distributed systems usually deal with issues such as heterogeneity and distribution (hence allowing the developer to focus his efforts on the functionality rather than the distribution). But we know that in case of mobile computing, these assumptions are not true. This is, firstly, due to the tremendous progress in development of mobile devices, a wide variety of devices are available, which also vary from one another in a variety of ways. On one hand, laptops offer relatively powerful CPUs and sufficient primary and secondary storage and on the other hand, cell phones have scarce resources and supplementing these resources is either expensive or impossible altogether. Secondly, in mobile systems, network connections generally have limited bandwidth, high error rates and frequent disconnections. Lastly, mobile clients usually have the ability to interact with various networks, services, and security policies as they move from one place to another.

Also, as the environment of a mobile device changes, the application behaviour needs to be adjusted to adapt itself to the changing environment. Hence dynamic re-configuration is an important building block of an adaptive system. Also, the interaction approach between the mobile client and the host dictates the effectiveness and efficiency of a mobile system. Asynchronous interaction deals with problems of high latency and disconnected operations that may arise with other interaction models. A client using asynchronous communication can issue a request and continue with its local operations and collect the output later. Hence the client and server modules are not required to execute concurrently to communicate with each other. Such an interaction permits reduction in bandwidth usage, decouples the client and server modules and improves the scalability of a system. Hence, given the highly variable computing environment of mobile systems, it is mandatory that modern middleware systems are designed that can support the requirements of modern mobile systems such as dynamic reconfiguration and asynchronous communication [Ka05].

In this paper, we present an architecture for a middleware named MAGI (Section 2), which enables heterogeneous mobile devices access to Grid services and implement an application toolkit that acts as a gateway to the Grid. This middleware provides support for delegation of jobs to the Grid, secure communication between the client and the Grid, offline processing, adaptation to network connectivity issues and presentation of results in a form that is in keeping with the resources available at the client device. We demonstrate (Section 3) that the addition of such a middleware causes minimum overhead and the benefits obtained by it outweigh this overhead.

2 Detailed Architecture

The MAGI middleware is exposed as a web service to the client applications. The components of the middleware (as shown in Figure 1) are discussed briefly as follows.

2.1 Discovery Service

The discovery of the MAGI middleware by mobile devices is managed by employing a UDDI (Universal Description, Discovery and Integration) registry [Ho02], [Lu04]. Once the middleware service is deployed and registered, other applications/devices would be able to discover and invoke it using the API in the UDDI specification [Lu04] which is defined in XML, wrapped in a SOAP envelope and sent over HTTP.

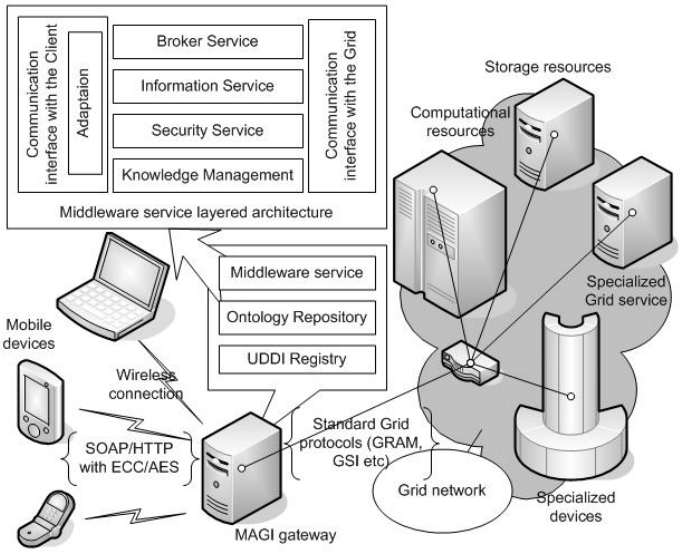

Figure 1: Deployment Model and Architecture of MAGI

2.2 Communication Interface with the Client

The interface advertised to the client is the communication layer between the mobile device and the middleware. This layer enables the middleware to operate as a web service and communicate via the SOAP framework [Do00].

2.3 Adaptive Management by Offline Processing

The advertisement of the MAGI middleware as a web service permits the development of the architecture in a manner that does not make it necessary for the client applications to remain connected to the middleware at all times while the request is being served.

This is done by utilizing *solicit-response* or *notification* operation types [Ce02]. We focus on providing software support for offline processing at the client device. For this we consider two kinds of disconnections; intentional disconnection, where the user decides to discontinue the wireless connection and unintentional disconnection, which might occur due to variation in bandwidth, noise, lack of power etc. This is made possible by pre-fetching information or meta-data only from the middleware service. This facilitates in locally serving the client application at the device. However, requests that would result in updates at the middleware service are logged so that they may be executed upon reconnection. To establish the mode of operation for the client application, a connection monitor is used to determine the network bandwidth and consequently the connection state (connected or disconnected). During execution, checkpoints are maintained at the client and the middleware in order to optimize reintegration after disconnection and incorporate fault tolerance.

2.4 Communication Interface with the Grid

The communication interface with the Grid provides access to the Grid services by creating wrappers for the API advertised by the Grid. These wrappers include the Globus framework protocols such as GRAM [Glo4], MDS [Cz01], GSI [We03] etc. which are mandatory for any client application trying to communicate with the Grid services. This enables the middleware to communicate with the Grid, in order to accomplish the job assigned by the client.

2.5 Broker Service

The broker service deals with initiating the job request and steering it on behalf of the client application. Firstly the client application places a request for a job submission. After determining the availability of the Grid service and authorization of the client, the middleware downloads the code (from the mobile device or from a location specified by the client e.g. an FTP/web server). Once the code is available, the broker service submits a "createService" request on the GRAM's Master Managed Job Factory Service (via the wrapper) which is received by the Redirector [Gl04]. The application code (controlled by the middleware) then interacts with the newly created instance of the service to accomplish the task. The rest of the process including creating a Virtual Host Environment (VHE) process and submitting the job to a scheduling system is done by GRAM. Subsequent requests by the client code to the broker service are redirected through the GRAM's Redirector.

A status monitor service (a subset of the broker service) interacts with GRAM's wrapper to submit FindServiceData requests in order to determine the status of the job. The status monitor service then communicates with the Knowledge Management module to store the results. The mobile client may reconnect and ask for the (intermediate/final) results of its job from the status monitor service.

2.6 Knowledge Management

All the data and information generated and processed by the different modules is handled by using semantic web technologies which provide the mechanisms to present the information as machine-processable semantics and building up the intelligent services to make decisions and perform knowledge level transformations on that information. It includes the management of user and device profiles, ontologies, policies, rule-bases, context information, system logs, performance metrics etc. The decisions and transformed information is then passed on to the relevant modules within the system or directly to the client or the Grid, which utilize it according to their specific needs. We can further use it purposefully for detection of conflicting goals of different modules and the resolution of those conflicts.

This module also maps each instance of a device's profile to different modes of operations $\{L_1, L_2, L_3, ..., L_n\}$ where each L_i is in accordance with the level of device's capabilities. The results are returned to the device according to the mapped mode of operation. The profiles (device specifications) are maintained at the ontology repository. If the device is not listed there, the client application submits it to the ontology repository otherwise it identifies its profile by sharing the URL of its ontology on the ontology repository (Figure 1). This information is fetched and processed and the reasoning service maps it to one of $\{L_1, L_2, L_3, ..., L_n\}$ or creates a new level of operation, L_{n+1}. Any results shown to the mobile device are scaled down or adjusted according to this mode of operation.

2.7 Information Service

This module interacts with the wrapper of the Globus framework's API for information services (MDS [Cz01]). It facilitates the client application by managing the process of determining which services and resources are available in the Grid (the description of the services as well as resource monitoring such as CPU load, free memory etc. Detailed information about Grid nodes (which is made available by MDS) is also shared on explicit request by the client.

2.8 Security

The Grid Security Infrastructure is based on public key scheme mainly deployed using the RSA algorithm [We04]. However key sizes in the RSA scheme are large and thus computationally heavy in context of handheld devices such as PDAs, mobile phones and smart phones etc. [Gu02]. We employ the Web Services Security Model [Gi02] to provide secure mobile access to the Grid. This web services model supports multiple cryptographic technologies. We use Elliptic Curve Cryptography (ECC) based public key scheme in conjunction with Advanced Encryption Standard (AES) for mobile access to Grid, which provides the same level of security as RSA and yet the key sizes are a lot smaller [Gu02].

This means faster computation with lower memory, bandwidth and power consumption accompanied with the same level of security as provided by the RSA public key encryption scheme. Communication between the user and middleware is based on security policies specified in the user profile. According to this policy, different levels of security can be used e.g. some users might just require authentication, and need not want privacy or integrity of messages. It may be noted that we emphasize on also providing security on Application layer, which also gives us the flexibility to change the security mechanisms if the need arises.

2.9 Communication between the Middleware Gateways

In case multiple instances of the MAGI middleware gateways are introduced for improving scalability, some problem scenarios might arise. Consider a mobile device that accesses the Grid network via gateway M_1, but disconnects after submitting the job. If the mobile device later reconnects at gateway M_2 and inquires about its job status, the system would be unable to respond if the middleware is not capable of sharing information with its other instances. This is achieved in the following manner.

We define a Middleware Directory Listing which maintains the ordered pairs (ID, URI) which will be used for the identification of the middleware instance. Also, we define an *X* service as a module of the middleware which facilitates in communication between any two middleware instances. After reintegration of the mobile client at M_2, *C* sends the ID of the middleware instance, where the job was submitted (i.e. M_1), to the *X* service. The *X* service determines that the ID is not that of M_2. The *X* service then checks the Middleware Directory Listing to find the URI corresponding to the Middleware instance M_1. The *X* service then requests (from the client application) the job-ID of the job submitted by *C*. Upon a successful response the *X* service communicates with the *X* service of M_1 using the URI retrieved. After mutual authentication, X-M_2 sends the job-ID along with the clients request for fetching the (intermediate/final) results to X-M_1. If the job is complete, the compiled results are forwarded to client application. In case the job isn't complete yet, the client application continues to interact with middleware service X-M_1 (where the job was submitted). Note that X-M_2 acts as a broker for communication between *C* and M_1. Also, if the C decides to disconnect and later reconnect at a third middleware instance M_3, then M_3 will act as a broker and communicate with M_1 on behalf of *C*. As all the processing of information is done at the middleware where the job was submitted, the other instances would only act as message forwarding agents.

3 Petri Net Model of the System

In this section we model the interaction between the mobile client and the MAGI middleware service. Our goal is to estimate the delay caused by the communication between the client and middleware service as well as the additional processing done by it. We use the time to completion of the whole process as an index of performance of our

middleware communication architecture. We keep the time taken by Grid processing constant as our results will be bench marked against it. The communication is modeled by using non-Markovian Stochastic Petri Nets [BPT00], [TB95].

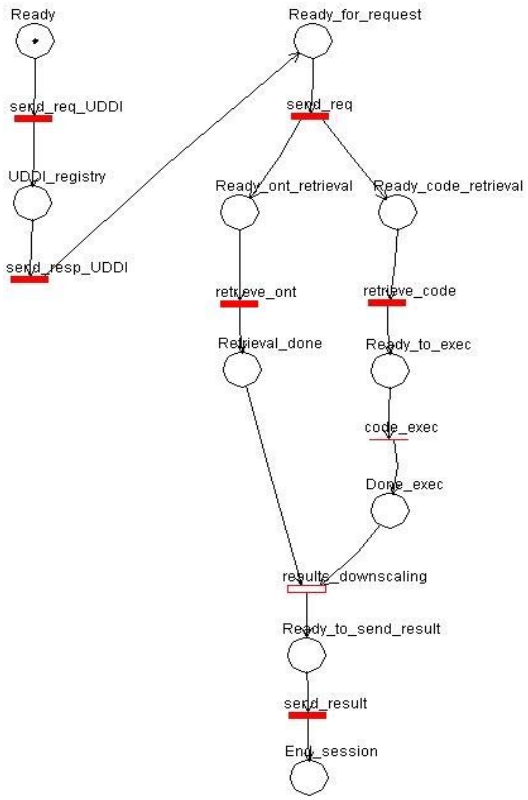

Figure 2: Petri Net model of the communication between mobile client and MAGI middleware

At first, the place *Ready* contains a token, indicating that the client device is ready to send a request. The *send_req_UDDI* transition is a timed transition modeling the transmission of request to the UDDI service. A token in the place *UDDI_registry* indicates that the request has been received by the UDDI service and it sends a response back to the client modeled by the timed transition *send_resp_UDDI*. At this point the client device is ready to send a request to the middleware indicated by a token in the place *Ready_for_request.* Next the transition *send_req* fires, which simulates the transmission of the job request to the middleware. Two arcs leave the *send_req* transition; one to the *Ready_ont_retrieval* place and the other to the *Ready_code_retrieval* place, indicating that the middleware service will perform two operations concurrently. One is retrieving the ontology (transition *retrieve_ont*) and the other is retrieving the code (transition *retrieve_code*). A token in *Retrieval_done* state shows that the ontology has been retrieved. The second token enters the *Ready_to_exec* state after the *retrieve_code* transition has been fired.

At this state the middleware is ready to execute the code. From here onwards until the results have been obtained, the Grid performs all the actions. We just simulate it as an immediate transition *code_exec,* as this time would be the same as in the case of a normal Grid user. When this transition fires, a token reaches the *Done_exec* place. At this point, the *result_downscaling* transition is enabled as there is already a token in *Retrieval_done* state. This transition models the process of scaling down the results according to the device's profile. After this transition fires, a token reaches the *Ready_to_send_result* place. The results are then sent to the client modeled by the *send_result* transition and finally its firing sends a token in the place *End_session* indicating the end of the session.

3.1 Parameters used in the Petri Net Model

To evaluate the Petri Net model in Figure 2, we used the following numerical parameters which are consistent with the ones used in [PRS01]. We give a description of the parameters and their assumed values as follows:

Table 1. Numerical values used for the parameters

Parameter	Description	Value
D_{req}	Dimension of client request	1 KB
$D_{\min}$	Minimum amount of Data	1 KB
$D_{\max}$	Maximum amount of Data	30 KB
D_{ont}	Dimension of ontology	10 KB
D_{req}	Dimension of client request	1 KB
D_{code}	Dimension of code	40 KB
λ_{scale}	Results scale down rate	4 requests/sec
Th_{high}	Throughput of wired network	1 Mbps
Th_{low}	Throughput of wireless network	10 Kbps – 1 Mbps

The firing rate of the *results_downscaling* transition has been fixed to λ_{scale} = 4 requests/sec. This factor is not only application dependent but also dependent on the computational power of the computer containing the middleware. However a value of 4 req/sec is a reasonable approximation as used in [PRS01].

Table 2. Parameters used in the Petri Net Model

Transition	Type	Expression
send_req_UDDI	Deterministic	D_{req} / Th_{low}
send_resp_UDDI	Deterministic	$D_{\min} / Th_{low}$
send_req	Deterministic	D_{req} / Th_{low}
retrieve_ont	Deterministic	D_{ont} / Th_{high}
retrieve_code	Deterministic	D_{code} / Th_{high}
code_exec	Immediate	-
results_downscaling	Exponential	λ_{scale}
send_result	Uniform	$[D_{\min}, D_{\max}] / Th_{low}$

We use the above mentioned parameters to describe the distributions associated with the transitions in the Petri Net model.

3.2 Results from the Petri Net Analysis

Based on the values in Table 1, we evaluated the Petri Net described in Figure 2 by using the WebSPN [Bo98], [Pu03] tool with which we can associate exponential as well as non-exponential firing rates to the transitions.

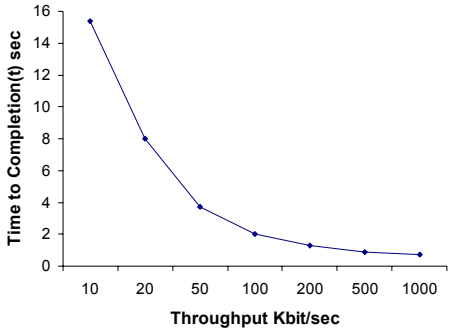

Figure 3: Time to completion (t) vs. throughput of the wireless network (Th_{low})

A value in the graph closer to the x-axis indicates lesser time taken by the middleware-client communication. As can be seen from the figure, with increasing throughput of the wireless network, the time to completion is reduced considerably and approaches zero as the throughput reaches 1 Mbps. The transition *send_result* takes a major portion of the overall time and hence the time to completion is dependent on the result size. Let's see the affect if we fix the result size to 1KB and keep the other values same. We can do that by making *send_result* a deterministic transition with firing rate $D_{\min} / Th_{low}$. After evaluating the Petri Net with this value, we obtain a graph shown in Figure 4.

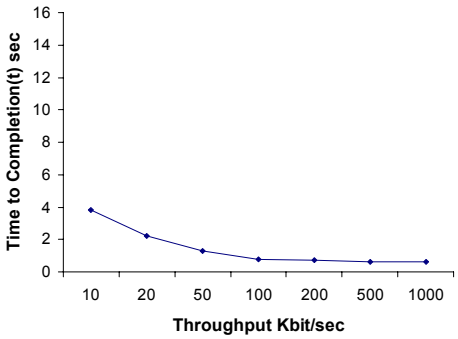

Figure 4: Time to completion (t) vs. Throughput of the wireless network (Th_{low}) with 1 KB result size

The graph shows no notable distinction with varying Th_{low}. We can conclude by studying the two graphs that the time to completion is affected by two parameters, namely the wireless network throughput and the result size. However, even with low throughput and considerably large result set, the time taken by the middleware to mobile device communication is within acceptable limits, only in the order of a few seconds in the worst case.

4 Related Work

Various efforts have been made to resolve similar issues. Signal [HA04] proposes a mobile proxy-based architecture that can execute jobs submitted to mobile devices, so in-effect making a grid of mobile devices. After the proxy server determines resource availability, the adaptation middleware layer component in the server sends the job request to remote locations. The efforts are more inclined towards QoS issues such as management of allocated resources, support for QoS guarantees at application, middleware and network layer and support of resource and service discoveries based on QoS properties.

In [Br01] a mobile agent paradigm is used to develop a middleware to allow mobile users' access to the Grid and it focuses on providing this access transparently and keeping the mobile host connected to the service. Though they have to improve upon the system's security, fault tolerance and QoS, their architecture is sufficiently scalable. GridBlocks [Gr04] builds a Grid application framework with standardized interfaces facilitating the creation of end user services. For security, they are inclined towards the MIDP specification version 2 which includes security features on Transport layer. They advocate the use of propriety protocol communication protocol and state that SOAP performance on mobile devices maybe 2-3 times slower as compared to a proprietary protocol. But in our view, proprietary interfaces limit interoperability and extensibility, especially to new platforms such as personal mobile devices.

5 Conclusions and Future Work

In this paper we identified the potential of enabling mobile devices access to the Grid. We focused on providing solutions related to distributed computing in wireless environments, particularly when mobile devices intend to interact with Grid services. An architecture for a middleware named MAGI is presented which facilitates implicit interaction of mobile devices with Grid infrastructure. Its interface is based on the web services communication paradigm. It handles secure communication between the client and the middleware service, provides for adaptive management via offline processing, manages the presentation of results to heterogeneous devices (i.e. considering the device specification) and deals with the delegation of job requests from the client to the Grid.

We also demonstrated that the addition of such a middleware causes minimum overhead and the benefits obtained by it outweigh this overhead. In future we intend to provide multi-protocol support in order to extend the same facilities to devices that are unable to process SOAP messages. Moreover, we will focus on trust issues, improving support for offline processing and incorporating self-management in the middleware through the knowledge management module [Sa05]. Along with this implementation we intend to continue validating our approach by experimental results.

References

[Bo98] Bobbio, A.; Puliafito, A.; Scarpa M.; Telek, M.: WebSPN: A WEB-accessible Petri Net Tool: International Conference on WEB based Modeling and Simulation, San Diego, California, 1998; pp. 137–142, 11–14.

[BPT00] Bobbio, A.; Puliafito, A.; Telek, M.: A modeling framework to implement preemption policies in non-Markovian SPNs: IEEE Transactions on Software Engineering, vol. 26, 2000; pp. 36-54.

[Br01] Bruneo, D.; Scarpa, M.; Zaia, A.; Puliafito, A.: Communication Paradigms for Mobile Grid Users: Proc. 10th IEEE International Symposium in High-Performance Distributed Computing, 2001.

[Ce02] Cerami, E.: Web Services Essentials: O'Reilly Publishers, Sebastopol, CA, USA, 2002; pp. 108-109.

[Cz01] Czajkowski, K.; Fitzgerald, S.; Foster, I.; Kesselman, C.: Grid Information Services for Distributed Resource Sharing: Proc. 10th IEEE International Symposium on High-Performance Distributed Computing (HPDC-10), IEEE Press, 2001.

[Do00] SOAP Framework: W3C Simple Object Access Protocol ver 1.1, World Wide Web Consortium recommendation: www.w3.org/TR/SOAP/

[FKT01] Foster, I.; Kesselman, C.; Tuecke, S.: The Anatomy of the Grid: Enabling Scalable Virtual Organizations: Int'l J. Supercomputer Applications, vol. 15, no. 3, 2001; pp. 200-222.

[Gi02] Giovanni, D.L. et al.: Security in a Web Services World; A Proposed Architecture and Roadmap: A joint security whitepaper from IBM Corporation and Microsoft Corporation, Version 1.0, April 2002.

[Gl04] GT3 GRAM Architecture: http://www-unix.globus.org/toolkit/docs/3.2/gram/ws/

[Gr04] GridBlocks project (CERN): http://gridblocks.sourceforge.net/

[Gu02] Gupta, V.; Gupta, S.; et al.: Performance Analysis of Elliptic Curve Cryptography for SSL: Proceedings of ACM Workshop on Wireless Security, Atlanta, USA, September 2002; pp. 87-94.

[HA04] Hwang, J.; Aravamudham P.: Middleware Services for P2P Computing in Wireless Grid Networks: IEEE Internet Computing vol. 8, no. 4, July/August 2004; pp. 40-46.

[Ho02] Hoschek, W.: Web service discovery processing steps, http://www-itg.lbl.gov/~hoschek/publications/icwi2002.pdf.

[Ka05] Kalim U.; Sajjad A.; Jameel, H.; Lee, S.Y.: Mobile-to-Grid Middleware: An Approach for Breaching the Divide Between Mobile and Grid Environments: Lecture Notes in Computer Science, Volume 3420, Jan 2005; pp. 1-8.

[Lu04] UDDI specification: www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm

[PRS01] Puliafito, A.; Riccobene, S.; Scarpa, M.: Which paradigm should I use? An analytical comparison of the client-server, remote evaluation and mobile agents paradigms': IEEE Concurrency and Computation: Practice & Experience, vol. 13, 2001; pp. 71-94.

[Pu03] WebSPN 3.2: http://campusone.unime.it/webspn/

[Sa05] Sajjad A. et. al.: AutoMAGI - towards an Autonomic middleware for enabling Mobile Access to Grid Infrastructure: International Conference on Autonomic and Autonomous Systems, Tahiti, 2005.

[TB95] Telek, M.; Bobbio, A.: Markov regenerative stochastic Petri nets with age type general transitions, Application and Theory of Petri Nets: 16th International Conference (Lecture Notes in Computer Science 935), Springer-Verlag, 1995; pp. 471–489.

[We03] Welch, V.; Siebenlist, F.; Foster, I. et al.: Security for Grid Services: HPDC, 2003.

[We04] Welch, V.; Foster, I.; Kesselman, C. et al.: X.509 Proxy Certificates for dynamic delegation: Proceedings of the 3rd Annual PKI R&D Workshop, 2004.