A Component-Based Architecture for an Autonomic Middleware Enabling Mobile Access to Grid Infrastructure

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

Department of Computer Engineering, Kyung Hee University, Giheung-Eup,
Yongin-Si, Gyeonggi-Do, 449-701, Republic of Korea
{ali, hassan, umar, yklee, sylee}@oslab.khu.ac.kr

Abstract. The increasing pervasiveness of wide-area distributed computing resources, like computational Grids, has given rise to applications that have inherent problems of complexity, adaptability, dynamism and heterogeneity etc. The emerging concept of autonomic computing holds the key to the self-management of such a multifarious undertaking and provides a way to further build upon this complexity without incurring additional drawbacks. Furthermore, access to Grid services at present is generally limited to devices having substantial computing, network and memory resources whereas most of mobile devices do not have the sufficient capabilities to be either direct clients or services in the Grid environment. The existing middleware platforms like Globus do not fully address mobility, yet extending the potential of the Grid to a wider audience promises increase in its flexibility and productivity. In this paper[1], we present a component-based autonomic middleware that can handle the complexity of extending the potential of the Grid to a wider mobile audience, by incorporating the features of context-awareness and self-management. We also address the middleware issues of job delegation to a Grid service, support for disconnected operation/offline processing and secure communication.

1 Introduction

Grid [1] computing permits participating entities connected via networks to dynamically share their resources. Its increasing usage and popularity in the scientific community and the prospect of seamless integration and interaction with heterogeneous devices and services makes it possible to develop further complex and dynamic applications for the Grid. However, 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 relatively reliable. For the same reason, the middleware technologies for such distributed systems encourage the developers to focus on the functionality rather than the distribution. But we know that

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

T. Enokido et al. (Eds.): EUC Workshops 2005, LNCS 3823, pp. 1225 – 1234, 2005.

in case of mobile computing, these assumptions are not essentially true. Firstly, there is a wide variety of mobile devices available; laptops offering substantial computing power and memory etc. to cell phones with scarce resources. 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 matters as they move from one place to another. So extending this potential of the Grid to a wider audience promises increases in flexibility and productivity, particularly for the users of mobile devices who are the prospective consumers and beneficiaries of this technology.

This goal was the main motivation behind the MAGI middleware [25], whose architecture provided the foundation support and infrastructure for building applications and services that provide mobile clients access to Grid. However, the efficient management of such a large platform is a considerably complicated issue and it is a constantly increasing complexity because of increasing numbers of heterogeneous devices and components being added to it. In fact, the current level of software complexity has reached such a level of complexity that it threatens the future growth and benefits of the general IT infrastructure [2]. Also, as the environment of a mobile device changes, the application/service behaviour needs to be adjusted to adapt itself to the new environment. Hence dynamic reconfiguration is an important building block of such an adaptive system. Furthermore, the interaction approach between the mobile client and the host dictates the effectiveness and efficiency of a mobile system. A promising approach to handle this complexity is the emerging field of autonomic computing [3]. It calls for the design of a system in such a way that it is aware of its constituting components and their details, it can reconfigure itself dynamically according to the change in its environment, optimize its working to achieve its goals and predict or recognize faults and problems in its work flow and rectify them. The inspiration of such self-managing approach has been taken from the autonomous nervous system and many efforts are underway for further development in this field [4], [5], [6]. The effect of such an autonomous system will be the reduction in the complexity and ease in management of a large system, and what better exemplar case for its application than the Grid and its middlewares. Various ways have been proposed to achieve the fulfillment of this vision, from agent-based [7] to policy-based self-management [8], from autonomic elements and closed control loops [9] to adaptive systems, self-stabilizing systems and many more, but the motivation and ultimate goal of all is the same. Hence, given the highly variable computing environment of mobile systems, it is mandatory that modern middleware systems are designed in such a way that they can support the requirements of modern mobile systems such as dynamic reconfiguration and asynchronous communication. The motivation behind the AutoMAGI middleware is to develop an autonomic middleware that provides mobile devices access to Grid infrastructure and also enables autonomous applications to use its platform. It will be beneficial to all kinds of Grid users, form the physicist who wants to run a set of simulations from his PDA to a doctor who wants a Grid medical service to analyze the MRI or CT scans of a patient from his smart phone, so that finally we can promise increase in seamless flexibility and productivity. In what follows, we first discuss the basic structure of autonomic components in our autonomic MAGI middleware and then present its architecture, which enables heterogeneous mobile devices to access Grid services and also enables autonomous applications to

use it as a platform for this purpose. This middleware provides support and management infrastructure for delegation of jobs to the Grid, a light-weight security model, offline processing, adaptation to network connectivity issues (disconnected operation) and presentation of results to client devices in keeping with their limited resources.

2 Autonomic Components in AutoMAGI

The AutoMAGI middleware is composed of autonomic components which are responsible for managing their own behavior in keeping with the relevant policies, and also for cooperating with other autonomic components to achieve their objectives. The structure of a typical component is shown in Figure 1.

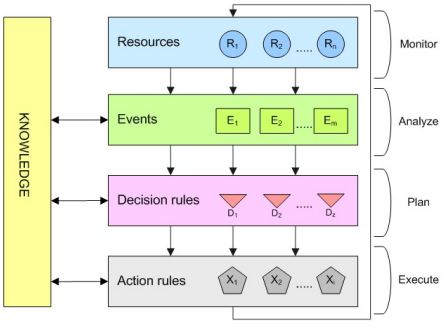

Fig. 1. Structure of an Autonomic Component in AutoMAGI

A resource can be anything, from a CPU, memory etc. to an application or service. The event signifies a change in the state of a resource. Decision rules are used for deducing a problem based on the information received from various events. The problems can be deduced using a single event or inferring from a group of events, based on a single occurrence or based on a history of occurrences. The component then makes plans to rectify this problem or optimize some functional/behavioral aspects, basing upon the policies and internal and external knowledge of the component. Action rules are then used to execute tasks to bring about these changes in line with the desired goal of the component.

This manner of structure facilitates in building up a control loop by employing the monitor, analyze, plan and execute cycle, which is of key significance for autonomic behavior [24].

3 AutoMAGI Architecture

The AutoMAGI middleware is exposed as a web service to the client application. The components of the middleware (as shown in Figure 2) are discussed briefly as follows.

3.1 Discovery Service

The discovery of the middleware by mobile devices is undertaken by employing a UDDI registry [11], [12]. The composition of the current web services may not give sufficient facilities to depict an autonomic behavior or to integrate them seamlessly with other autonomic components but with the advent of semantic web service technologies like OWL-S [13], it becomes possible to provide a fundamental framework for representing and relating devices and services with their policies and describing and reasoning about their functionalities and capabilities. Another hurdle is that the current organization and management of Web services and Grid services are static and must be defined a priori. However, by using Web Services Distributed Management (WSDM) [26], we get a mechanism of managing resource-oriented and dynamic web services and their discovery.

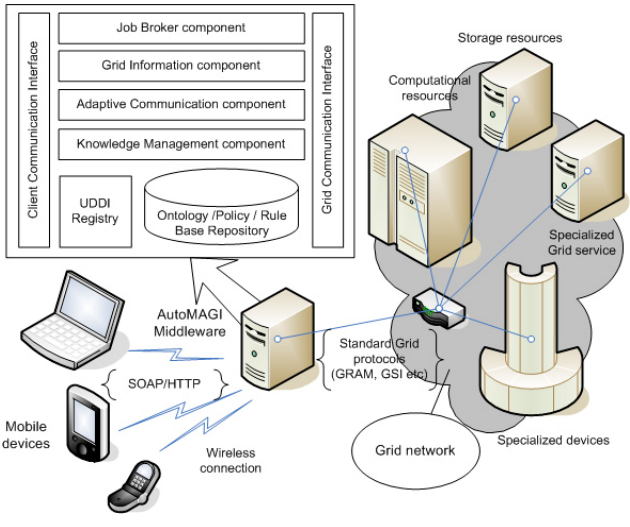

Fig. 2. AutoMAGI – Architecture and Deployment diagram

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 [12] which is defined in XML, wrapped in a SOAP envelope and sent over HTTP.

3.2 Client Communication Interface

The service advertised to the client is the communication interface between the mobile device and the middleware. This layer enables the middleware to operate as a semantic web service [14] and communicate via the SOAP framework [15]. The repository in the middleware also contains the device and user related ontologies and service policies for the devices, users and applications. Due to this service-oriented approach, it is not expected of the client to remain connected to the middleware at all times while the request is being processed.

3.3 Adaptive Communication Component

We focus on providing support for offline processing in the Adaptive Communication component, which allows the system to automatically redeploy essential components to maintain service availability, even in adverse circumstances. The adaptation scheme used is application aware, i.e., the framework of the Adaptive Communication component allows the developer to determine the application policies, which describe how the component will react in different situations. This enables the component to reconfigure itself while adapting to the prevailing conditions. With an autonomic policy manager to direct these reconfigurations, the services can maintain their availability even in disconnected mode. Two kinds of disconnections are considered for providing offline processing; voluntary and involuntary disconnection. The former refers to a user-initiated event that enables the system to prepare for disconnection, and the latter to an unplanned disconnection (e.g., due to network failure). The difference between the two cases is that in involuntary disconnection the system needs to be able to detect disconnection and reconnection, and it needs to be pessimistically prepared for disconnection at any moment, hence requiring to proactively reserve and obtain redundant resources (if any) at the client. The Adaptive Communication component utilizes a connection monitor for this purpose but the job for facilitating in making such decisions that whether the disconnection was intentional or not is done with the help of decision rules of the component. The action rules of the component take the contributing factors and parameters into account and based on the particular policy, proceed with the course of actions to be undertaken.

The flow of events dealing with the disconnected operation can be seen in the state-transition diagram in Figure 3. During execution, checkpoints are maintained at the client and the middleware, taking into account the policies of the device, user and application, in order to optimize reintegration after disconnection and incorporate fault tolerance.

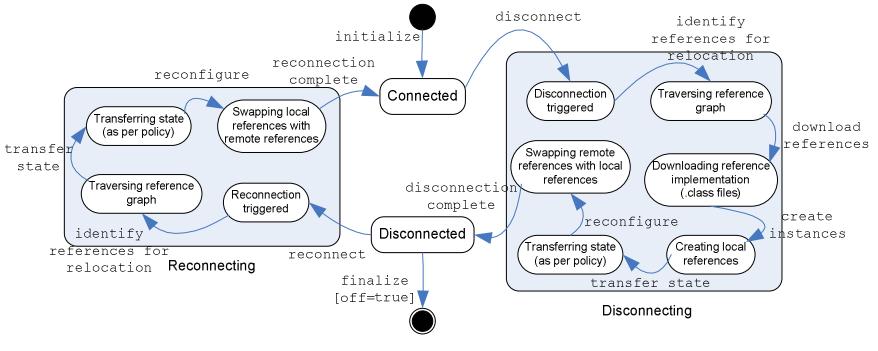

Fig. 3. State-transition diagram from disconnection/reconnection management

3.4 Grid Communication Interface

The Grid Communication interface provides access to the Grid services by creating wrappers for the API advertised by the Grid. These wrappers include standard Grid

protocols such as GRAM [16], MDS [17], GSI [18] etc. which are obligatory for any application trying to communicate with the Grid services using Globus. This enables the middleware to communicate with the Grid, in order to accomplish the job assigned by the client. Its working is in close collaboration with the Grid Information component, which is discussed later in the section.

3.5 Grid Information Component

The Grid Information component interacts with the wrapper of the GLOBUS toolkit's API for information services (MDS [17]). It assists 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.

3.6 Job Broker Component

The autonomic Job Broker component deals with initiating the job request and steering it on behalf of the client application. After going through the related policy and determining the availability of the Grid service and authorization of the client, it 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 Job Broker component submits a "createService" request on the GRAM's Master Managed Job Factory Service (via the wrapper) which is received by the Redirector [16]. The application code (controlled by the application policy) 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 Job Broker component are redirected through the GRAM's Redirector.

The monitoring service of the Job Broker component interacts with GRAM's wrapper to submit FindServiceData requests in order to determine the status of the job. It may then communicate with the Knowledge Management component to store the results, depending on the type of application and all the related policies, as the mobile client may reconnect and ask for the results (intermediate/final) of its job from the Job Broker component.

3.7 Knowledge Management Component

The knowledge used by different autonomic components is handled by using semantic web technologies in the middleware which provide the mechanisms to present the information as machine-processable semantics and is useful in building intelligent decision-making mechanisms and perform knowledge level transformations on that information. These decisions and transformed information is then passed on to other components within the system or directly to the client or the Grid, which utilize it according to their specific needs.

The autonomic components that constitute the AutoMAGI middleware constantly monitor and gather the data they need to react to or act upon, according to their management tasks and targets. This wide-scoped data is elaborated and organized through

the notion of events. Events in turn are typically meaningful in a certain context when related with other events. This correlation information can then be used for Data filtering, measuring thresholds and sequencing of actions etc. But for such an autonomic model to work properly, a shared knowledge must be present which includes features context information, system logs, performance metrics and relevant policies. We manage this knowledge base with the help of a Policy Manager in the KM component. Due to the autonomic character of the Knowledge Management component, the middleware is able to respond to a problem that happened in the defined problem space (scope of defined problems) and use predictive methods to discover probable problems in advance and so succeed in achieving better results and eliminating problems. But as each autonomic element has its own knowledge model, the problem of data/knowledge integration might result, which is again handled by the Knowledge Management module. Furthermore, some conflict-scenarios may arise due to the conflicting goals pursued by different autonomic components. The optimal solution of such conflicts is also the job of this component.

3.8 Security Component

The Grid Security Infrastructure is based on public key scheme mainly deployed using the RSA algorithm [19]. However key sizes in the RSA scheme are large and thus computationally heavy on handheld devices such as PDA's, mobile phone's, smart phones etc. We propose the use of Elliptic Curve Cryptography (ECC) based public key scheme, which can be used in conjunction with Advanced Encryption Standard (AES) for mobile access to Grid. This provides the same level of security as RSA and yet the key sizes are a lot smaller [20] which means faster computation, low memory and bandwidth and power consumption with high level of security.

Furthermore, as no autonomic element should provide its resources or services to any other component without the permission of its manager, we make use of security policies that govern and constrain their behavioral aspects at a higher level. The security policies include different characteristics like the level of protection needed to be applied to the various information resources that the component contains or controls, rules that determine how much trust the element places in other elements with which it communicates, cryptographic protocols the element should use in various situations and the circumstances in which the element should apply or accept security-related patches or other updates to its own software, and so on. Each autonomic component also holds various security related tasks and state representations to describe the current status and activities, like level of trust on other communicating entities, notification form other components or human administrators of suspicious circumstances, agreements with other components regarding provision of security-related information, such as log-file analyses or secure time stamping, and a list of trustworthy resource suppliers (used to quickly verify the digital signatures on the resources they provide).

4 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 other instances. To manage resources, clients and requests etc. between themselves, the distributed instances of AutoMAGI middleware use an Arbiter component. So in accordance with high-level guidance from the application/client's policies for the functional environment and the load-balancing policies from the middleware, we attain a guideline for optimal sharing of knowledge between different middleware instances.

The Arbitrator facilitates in communication between any two middleware instances. It maintains the ordered pairs (ID, URI) which are used for the identification of the middleware instance. So for instance, 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 Arbitrator. The Arbitrator determines that the ID is not that of M_2. It then checks the Middleware Directory Listing to find the URI corresponding to the Middleware instance M_1. The Arbitrator then requests (from the client application) the job-ID of the job submitted by C. Upon a successful response the Arbitrator of M_2 (A-M_2) communicates with the Arbitrator of M_1 (A-M_1) using the URI retrieved. After mutual authentication, A-M_2 sends the job-ID along with the clients request for fetching the (intermediate/final) results to A-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 A-M_1 (where the job was submitted). Note that A-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 A-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.

5 Related Work

Signal [21] 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 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 [22] 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 improvement is needed in the system's security, fault-tolerance and QoS, the architecture is sufficiently scalable. GridBlocks [23] 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 communication protocols based on the statement that performance of SOAP on mobile devices is 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 and certainly an autonomic computing system will only be possible if open standards are ensured.

6 Conclusions and Future Work

In this paper we identified the potential of enabling mobile devices access to the Grid and how we use the emerging autonomic computing paradigm to solve the management complexity. The component-based architecture of an autonomic middleware named AutoMAGI is presented which facilitates implicit interaction of mobile devices with Grid infrastructure. It ensures secure communication between the client and the middleware service, provides support for offline processing, manages the presentation of results to heterogeneous devices considering the device specifications and deals with the delegation of job requests from the client to the Grid.

In future we intend to focus on issues of autonomic security (self-protection) and streamline the Knowledge Management component for self-optimization. Along with a prototype implementation, we intend to continue validating our approach by experimental results and benchmarks.

References

1. 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) 200-222
2. Wladawsky-Berger, I.: Advancing E-business into the Future: The Grid. Kennedy Consulting Summit. New York (2001)
3. Ganek, A.G., Corbi, T.A.: The dawning of the autonomic computing era. IBM Systems Journal, v.42 n.1 (2003) 5-18
4. Horn, P.: Autonomic Computing: IBM's Perspective on the State of Information Technology. http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf: IBM Corporation (2001)
5. HP Adaptive Enterprise strategy: http://www.hp.com/go/demandmore
6. Microsoft Dynamic Systems Initiative: http://www.microsoft.com/windowsserversystem/dsi/default.mspx
7. Bonino, D., Bosca, A., Corno, F.: An Agent Based Autonomic Semantic Platform. First International Conference on Autonomic Computing. New York (2004)
8. Chan, H., Arnold, B.: A policy based system to incorporate self-managing behaviors in applications. Companion of the 18th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications. (2003) 94-95
9. Kephart, J., Chess, D.: The Vision of Autonomic Computing. Computer, v.36 n.1 (2003) 41-50
10. 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) 71-94
11. Hoschek, W.: Web service discovery processing steps. http://www-itg.lbl.gov/~hoschek/publications/icwi2002.pdf
12. UDDI specification: http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm
13. OWL-S: OWL-based Web Service Ontology: http://www.daml.org/services/owl-s/
14. Semantic Web Services Initiative (SWSI): http://www.swsi.org/
15. SOAP Framework: W3C Simple Object Access Protocol ver 1.1, World Wide Web Consortium recommendation. 8 May 2000; www.w3.org/TR/SOAP/
16. GT3 GRAM Architecture: www-unix.globus.org/developer/gram-architecture.html

17. Czajkowski, K., Fitzgerald, S., Foster, I., Kesselman, I.: Grid Information Services for Distributed Resource Sharing. Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing, IEEE Press (2001)
18. Welch, V., Siebenlist, F., Foster I., et al.: Security for Grid Services. HPDC (2003)
19. 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)
20. Gupta, V., Gupta, S., et al.: Performance Analysis of Elliptic Curve Cryptography for SSL. Proceedings of ACM Workshop on Wireless Security. Atlanta, GA, USA (2002) 87-94
21. Hwang, J., Aravamudham, P.: Middleware Services for P2P Computing in Wireless Grid Networks. IEEE Internet Computing vol. 8, no. 4 (2004) 40-46
22. Bruneo, D., Scarpa, M., Zaia, A., Puliafito, A.: Communication Paradigms for Mobile Grid Users. Proceedings 10th IEEE International Symposium in High-Performance Distributed Computing (2001)
23. GridBlocks: Helsinki Institute of Physics, (CERN). http://gridblocks.sourceforge.net/docs.htm
24. Herrmann, K., Mühl, G., Geihs, K.: Self-Management: The Solution to Complexity or Just Another Problem? IEEE Distributed Systems Online, vol. 6, no. 1 (2005)
25. Sajjad, A., Jameel, H., et al.: MAGI - Mobile Access to Grid Infrastructure: Bringing the gifts of Grid to Mobile Computing. Proceedings of 2nd International Conference on Grid Service Engineering and Management. Erfurt, Germany (2005)
26. Web Services Distributed Management (WSDM) standards specifications: http://docs.oasis-open.org/wsdm/2004/12/wsdm-1.0.zip/