An Interceptor Model to Provide Dynamic Adaptation in the InteGrade Grid Middleware

Jesus José de Oliveira Neto, Fábio M. Costa

[1]Institute of Computing – Federal University of Goiás
Campus 2 - UFG, 74690-815 - Goiânia-GO, Brazil

{jesus,fmc}@inf.ufg.br

Abstract. *Computer grids typically combine computational resources of diverse types, such as storage, processing and scientific instruments, spread across different administrative domains. As a result, computer grids are a complex and diversified execution environment, which exhibit high variation of resource availability and node instability, among other problems. This paper presents a model for dynamic interceptors and its use in InteGrade, an opportunistic grid middleware. The use of interceptors aims to provide dynamic adaptation capabilities to InteGrade through its underlying communications middleware, contributing to make InteGrade able to deal with the highly variable execution environment of opportunistic grids. The aim is to show that existing dynamic adaptation capabilities, a common feature of modern distributed systems middleware, can be used to provide a significant level of flexibility to grid computing middleware, mostly displacing the need to (re-)implement specific adaptation mechanisms for the latter kind of middleware.*

1. Introduction

The wide availability of middleware technologies, used to support numerous applications and services, resulted in considerable maturity in this area. In particular, applications that need to adapt to variations in the execution environment and in the user/device's context have become more common. Mobile and distributed multimedia applications are well-known examples of this trend. Such applications demand a high level of flexibility and adaptability from the middleware platform, which must be able to self-adjust themselves in response to dynamically changing application requirements.

More recently, computing grids [Foster and Kesselman 2004] have appeared as another example of distributed computing environment with a high degree of dynamism. This is particularly true regarding opportunistic grids, where resource availability is constantly changing. Appropriate, flexible, middleware support, matching such dynamism, is an essential feature to enable the development of effective grid computing applications.

This paper presents an approach to provide such flexibility to grid computing middleware, by means of an interceptor model present at the level of the underlying communications middleware. Such interceptors enable runtime adaptation of the behavior of both middleware and application components. We argue that the adaptation features that have become common in distributed systems middleware are sufficient for most of the flexibility needs of grid middleware, which can use those features as the building blocks to compose sophisticated dynamic adaptation features. This obviates the need for adaptation

mechanisms that are specific for grid computing middleware, further leveraging existing research on adaptable and reflective middleware.

The interceptor model was applied in InteGrade [Goldchleger et al. 2004], an opportunist grid middleware with no previous support to deal with the high dynamism that occurs in computer grids. Interceptors were implemented in OiL [Maia et al. 2005], a highly dynamic middleware that serves as one of InteGrade's underlying communications middleware platforms. Using such interceptors, the components of InteGrade can be adapted at runtime, effectively changing the grid's overall behavior. The approach has been demonstrated by means of two applications: dynamic deployment of firewall traversal mechanisms and dynamic adaptation of the grid's resource management protocol.

This paper is organized as follows. Section 2 presents the InteGrade grid middleware and its main components. Section 3 describes the main characteristics of the dynamic interceptor model. Section 4 presents two applications of the interceptor model in InteGrade, while Section 5 discusses related work. Finally, Section 6 presents the conclusion and future work.

2. InteGrade - Opportunistic Grid Middleware

InteGrade [Goldchleger et al. 2004] is an opportunistic grid middleware with support for the development of applications that use idle resources available in computer networks. It uses two different CORBA-compliant ORBs, JacORB [Spiegel 2005] and OiL [Maia et al. 2005], as communications middleware to abstract away the details of remote interaction among its components. These ORBs allow the integration of components that have been implemented in different languages, running on diverse hardware and software platforms. In particular, the design of InteGrade requires support for the integration of components written in Java, C/C++ and Lua. The main components of the InteGrade architecture are:

- Application Submission and Control Tool (ASCT): a graphical user interface that allows users to submit applications and control their execution;
- Application Repository (AR): stores the code of applications that are executed on the grid nodes;
- Local Resource Manager (LRM): runs on each grid node and is responsible for loading and executing applications scheduled for that node. It is also responsible to inform GRM about local resource availability;
- Global Resource Manager (GRM): manages the resources on a cluster; it receives notifications of resource availability from the LRMs (through an information update protocol), and runs a scheduler that allocates tasks to nodes based on resource requirements and availability;
- Execution Manager (EM): keeps information about each running application, such as its state, executing nodes, input and output parameters. It also coordinates the application recovery process in case of failures.

3. The Dynamic Interceptor Model

Interceptors are a mechanism commonly used to extend the basic middleware functionality, modifying its behavior regarding non-functional properties. They can be used to insert and configure non-functional properties, such as fault tolerance and quality of service, in

a transparent way, without changing the middleware architecture. There are two ways to activate interceptors in a middleware platform:

- Static: the decision to use interceptors is made at compile time, before initialization of the platform. The Portable Interceptors defined in the CORBA specification [OMG 2008] are an example of this.
- Dynamic: the decision is made at runtime, thus enabling interceptors to be activated after initialization of the platform. CORBA's portable interceptors can also be activated in this way, although their installation must be decided prior to the beginning of ORB execution.

Our interest in this paper, however, is on interceptors that can be installed and made active at runtime, even though their use was not decided a priori.

3.1. Overall Approach

In this work, a dynamic interceptor model was developed at the communications middleware level to demonstrate its feasibility and benefits to provide dynamic adaptation capabilities for the grid middleware.

Although InteGrade's focus is on opportunistic grids, which are highly dynamic environments, it is not currently capable of dynamic self-reconfiguration to deal with the frequent changes that occur on the grid, such as the availability of resources and network connectivity.

Figure 1 shows the approach followed in this work to circumvent this limitation. Initially, InteGrade is seen as a static grid middleware that uses two underlying ORBs (JacOrb and OiL) for the communications among its internal components. The dynamic interceptor model is then introduced in the two ORBs. Using this dynamic adaptation mechanism, the ORBs will be capable of manipulating their non-functional properties to self-adjust to the changes that occur in the execution environment. Through this dynamic interceptor model, InteGrade will be able to dynamically modify the way as its components interact. This means that InteGrade will be able to self-adjust, at runtime, according to changes that occur in the non-functional requirements of its components. For instance, this could be used to transparently create replicas of an application's tasks for fault-tolerance, by intercepting messages of the application execution protocol used by the GRM component.

It is necessary to note that the dynamic interceptor model is not sufficient to make InteGrade capable of structural adaptation (through the insertion, replacement or removal of components). For instance, interceptors cannot be used to adapt the platform with new functional features, such as by replacing the implementation of the component in charge of task scheduling (GRM). Nevertheless, such features could be made available, using the same overall approach advocated in this paper, by adopting architectural adaptation mechanisms (as a complement to interceptors) if they are provided by the underlying communications middleware.

3.2. Architeture

The architecture of the dynamic interceptor model is based on the *Interceptor* design pattern [Schmidt et al. 2000], being composed of three components. The first component

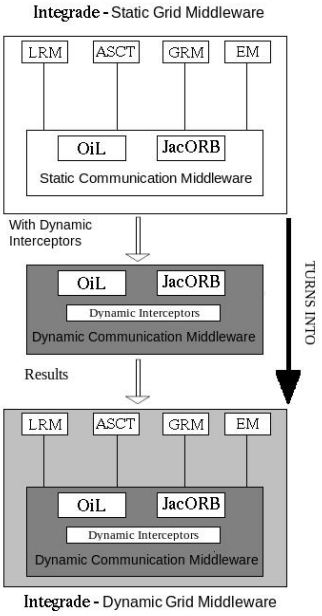

Figure 1. Overall approach

is the *Monitor*, responsible for inspecting the execution environment and the internal state of the ORB. In case some change occurs, either in the execution environment or inside the ORB, the *Monitor* will decide which non-functional property must be used. When the *Monitor* is called, a component named *Context*, described below, must be passed as parameter. The *Monitor*, in turn, must forward this component to the *Implementor*. The second component is the *Implementor*, which contains the implementation of mechanisms that provide a given a non-functional property, which will be applied in response to some change detected by the *Monitor*.

The third component is the *Context*, which allows the *Monitor* to have access to internal information of the ORB. The component creation occurs when an interception point inside the ORB is reached. The information type obtained from the *Context* depends on the event that drove the ORB to invoke the *Monitor* at this point. This event can be, for example, a client request being sent to a remote object, or a reply coming from this same remote object. Thus, the *Monitor* would have information about the requests and/or replies that had been sent and received. This information can be used by the *Implementor* components to configure a non-functional property.

Figure 2 presents the architecture of the interceptor model. The *Monitor* can have access to one or more *Implementor* components, each one implementing a given non-functional property. The *Context* provides information about the ORB's internal state at the point where the interceptor is inserted; this information is accessed by the *Monitor* and then forwarded to the *Implementor*. When changes occur inside the ORB or in the execution environment, the *Monitor* loads one of its *Implementor* components to adequately deal with them.

An instance of the interceptor model can be inserted at different points inside the ORB. At each interception point there is only one instance of the interceptor model. The interception points inside the ORB invoke an interceptor whenever some type of event

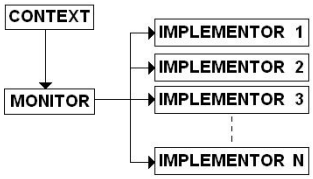

Figure 2. Architeture of the Dynamic Interceptor Model

occurs. The interceptor then checks if there is need to apply a non-functional property. The definition of the point inside the ORB where a non-functional property must be applied by the interceptor depends on the type of information about the ORB's internal state available at each interception point.

For example, for the introduction of fault-tolerance through replication, this non-functional property must be put in the interception point of the ORB where it is possible to have access to the requests that are being created. In an ORB client based on the CORBA specification, this point would be between the *Stub* and the *ORB Core*. Thus, the interceptor could replicate the requests and send them to all the remote replicas of the object. If the original object fails, the interceptor would use the results provided by some of the replicas.

3.3. Implementation

The dynamic interceptor model was implemented in the Lua language [Ierusalimschy 2006] as an integral part of OiL. Using Lua's reflective features, the interceptor model is capable to dynamically activate non-functional properties by loading and activating interceptor code at runtime.

In Figure 3, we show an example of sending a client's request, passing through the main components of OiL, until the request arrives at the server, and later following the reverse path for the server's reply. Along this path, both for request and reply messages, interception points are implemented enabling adaptation of the mechanisms used to handle them.

As mentioned in the previous section, the points inside the ORB where non-functional properties must be applied is determined by dynamic interceptors based on information about the ORB's internal state in those points. The interception points shown in Figure 3 are described below:

- The *Send_Request* and *Receive_Reply* points on the client side and the *Send_Reply* and *Receive_Request* points on the server side are used to have access to request parameters and results.
- The *Access_IOR_Socket* point inside the *GIOP/IIOP Client* component is used to have access to an IOR [OMG 2008] obtained by OiL on the client side. This point also allows access to the socket used to connect to the ORB server.
- The *Create_IOR* point inside the *GIOP/IIOP Server* serves to access the IORs generated by OiL on the server side.
- The *Output_Stream_Client* and *Input_Stream_Client* points on the client side and the *Output_Stream_Server* and *Input_Stream_Server* points on the server side can be used to access the data stream sent and received via the network.

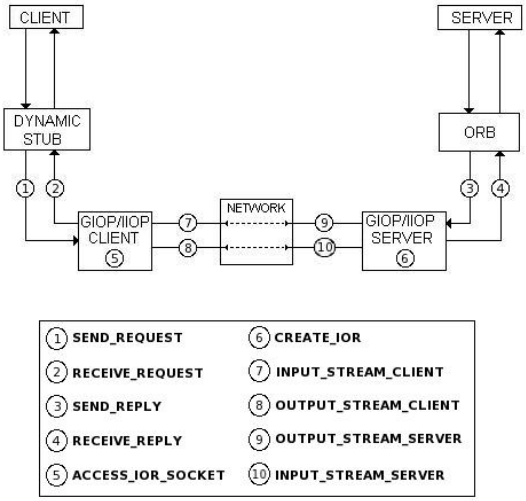

Figure 3. Interception Points inside OiL

4. Applications Cases for the Interceptor Model

4.1. Firewall/NAT Traversal

The resources and services provided by computer grids generally are distributed throughout multiple administrative domains, present in several locations around the world. Due to this high dispersion, often there are diverse academic or corporate institutions that are responsible for the domains that constitute a grid.

The policies and restrictions defined in these administrative domains have the goal to provide greater control and security over resources and services. Firewalls are an example of this. However the use of firewalls limits the connectivity of computers on the network. This limitation hinders the capability of grid middleware systems to integrate resources from different administrative domains.

Other mechanism frequently used in many administrative domains and which can hinder grid middleware is Network Address Translation (NAT). This service creates a private IP address space inside the internal network and forbids these addresses to be accessed from the external network. When a computer in this internal network has to send external messages, the NAT server converts its local address to some public address belonging to the administrative domain of the institution [OMG 2004].

In InteGrade, this problem occurs, for example, when a parallel application is executed by a set of nodes on a grid belonging to different organizations. These nodes, in order to communicate with each other, need to know their respective addresses, but if some of them happen to be on a NAT network, their internal addresses will not have external meaning, since they are not public addresses, hindering access from external networks.

To deal with these problems, the OMG approach for NAT/Firewall traversal [OMG 2004] was implemented through the dynamic interceptor model, making possible for InteGrade components to communicate even though they might be on distinct networks protected by such devices.

However, if we had strictly followed the OMG specification to implement firewall and NAT traversal, an ORB client would need a number of modifications to be capable of knowing when connecting to one or more application proxies, thus knowing whether the server objects are behind a protected network, which undermines transparency. An ORB server, in a protected network with firewall and NAT, also would have to be modified to inform that its external access must be made through a proxy.

In contrast, the use of dynamic interceptors allows the OMG approach to be implemented in a ORB with only small changes in the code of the client and server-side components, in a way that is largely transparent to client and server objects. These changes basically consist in the inclusion of simple calls for the *Monitor* component at different points of the request-reply path. Importantly, the firewall and NAT traversal feature is loaded and activated only at runtime and when needed, a feature that is natural when using the interceptor model.

4.1.1. Typical Scenario

Figure 4 illustrates the use of firewall and NAT traversal in InteGrade using dynamic interceptors. In this example, one of the LRMs is on network A, which does not have firewall and NAT. This LRM tries to connect to the *Cluster Manager* (GRM), which is located on another network, B, protected by firewall and NAT.

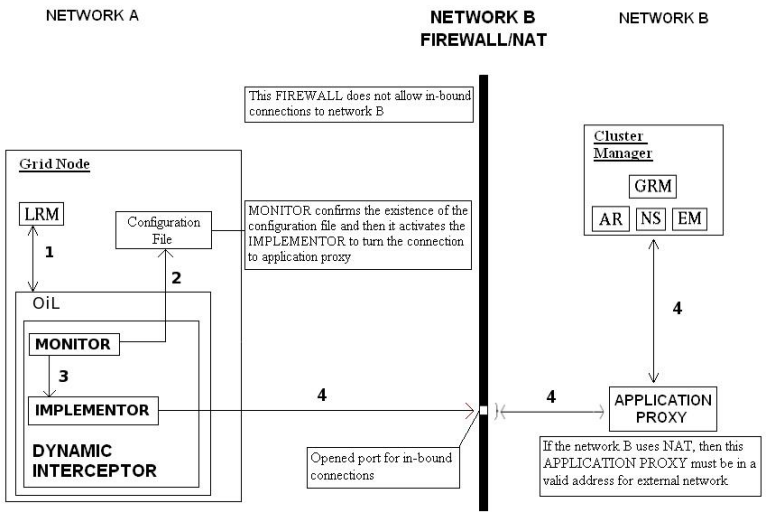

Figure 4. Using the Interceptor Model for firewall/NAT traversal.

The steps for LRM to connect to the *Cluster Manager* via an application proxy in this scenario are described below:

1. LRM tries to connect to the *Cluster Manager* through OiL's services, as usual;
2. the *Monitor* component searches for a configuration file that contains the application proxy address;
3. after verifying the existence of the configuration file, the *Monitor* activates the *Implementor* that deals with firewall/NAT traversal;

4. next, the *Implementor* redirects the connection to the application proxy. As shown in the figure, the listening port used by the proxy must be open on the firewall to receive external connections. Only then the LRM is able to communicate with the proxy. Note that for this scheme to work with NAT, the application proxy must be on a machine with a valid IP address.

The implementation of firewall/NAT traversal in this work served as a means to demonstrate the use of the interceptor model rather than an end in itself. The goal was to show that the traversal mechanisms can be dynamically deployed. Nevertheless, there is still the need to configure an open port on the firewall by some external means. Removing this limitation could be the subject for further work.

4.1.2. Experimental Results

In this section, we provide evaluation results obtained using the interceptor model for a simplified version of the firewall traversal case. The environment used for the tests is the same described in the previous subsection. Thus, the *Cluster Manager* and application proxy had been instantiated on two machines that are on a network protected behind a firewall. However, this network does not use NAT. The LRM was placed on the third machine that is part of a common network without firewall. Table 1 details the hardware and software characteristics used for the experiments.

Table 1. Description of the experimental environment

Processor	Pentium IV 3.2 GHz HT
Memory RAM	1024 MB
Hard Disk	160 GB
Network Interface	Realtek 8169
Operating System	Linux Slackware Version 11.0.0 i386 Kernel 2.6.18
InteGrade	Version 0.3-RC1
OiL	Version 0.3.1-Alpha
Lua Language	Version 5.0.2

The network topology used for the tests is shown in Figure 5. As we can see, the machines of network A are connected with the machines of network B through a switch. However, between this switch and network B machines, there is a firewall.

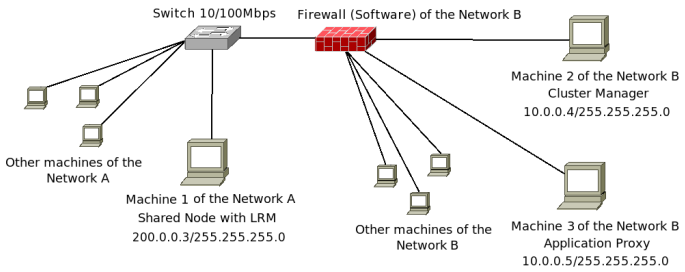

Figure 5. Network Topology used in the experimental tests

The measurements of the overhead of the interceptor model were obtained through the analysis of two LRM common operations: inicialization and the request for execution of a simple application. In each one of these operations, three tests were conducted, as described below.

- Test I: Execution of an operation in the LRM without dynamic interceptors inserted in the ORB.
- Test II: Execution of an operation in the LRM with the dynamic interceptor model loaded, but with no extra non-functional property, i.e., with no active interceptors.
- Test III: Execution of an operation in the LRM with the dynamic interceptor model and also with the firewall and NAT traversal implemented through an application proxy.

The test results for the LRM initialization operation are shown in Table 2. For each test, 50 calls were made. This table shows the maximum time, the mininum time and the average time for the LRM initialization. The measurement was taken in milliseconds (ms).

Table 2. *Overhead* Measurement for LRM Initialization

	Maximum Time	Average Time	Mininum Time
Test I	820 ms	810 ms	800 ms
Test II	880 ms	850 ms	830 ms
Test III	910 ms	860 ms	840 ms

In its turn, the test results for requesting the execution of a simple application are shown in Table 3. For each one of these tests, 50 calls were made and, in the same way, this table shows the maximum time, the mininum time and the average time for execution of a simple application.

Table 3. *Overhead* Measurement for requesting a execution of a simple application (in ms)

	Maximum Time	Average Time	Mininum Time
Test I	20 ms	10 ms	0 ms
Test II	20 ms	10 ms	0 ms
Test III	50 ms	20 ms	0 ms

From these experiments, we observed that there was an increase in the initialization time for LRM when the dynamic interceptors model is inserted. With firewall and NAT traversal, the initialization time increased a little more.

Table 2 shows that the increase caused by the interceptor model and also by the implementation of the firewall and NAT traversal was small. For example, in the third test, the increase in the average initialization time was only 6% in relation to the first test.

According to Table 3, the mere presence of the interceptor model alone did not cause a time increase for requesting the execution of a simple application, as shown in the second test. However, there was a larger time increase when the application proxy is used for firewall and NAT traversal. Although this increase was high, the overall execution time for this operation was kept relatively small. Thus, we observe that the increases caused by the interceptor model and the firewall and NAT traversal implementation cause a small impact in the performance of InteGrade.

Nevertheless, if the interceptor model and the firewall/NAT traversal implementation are used in an large scale environment, with some LRMs trying to communicate with a *Cluster Manager* placed on a protected network, the network performance will possibly

decrease. This situation can happen if only one application proxy is responsible for data transmission. Thus, this proxy will cause a ”bottleneck” in the network. To prevent this, replicated application proxies can be provided.

4.2. Frequency of Resource Usage Update Messages

InteGrade is an opportunistic grid middleware and, therefore, must be constantly informed of resource availability on the machines that compose the grid. Part of this information, such as the amount of free memory, varies throughout the grid's lifetime.

Thus, the LRMs on the machines that share resources with the grid need to inform GRM about resource availability on a regular, periodic, basis, as well as when there are sudden significant changes.

The frequency of such updates must be determined according to a tradeoff. In case the frequency is too high, there will be a significant communications cost, but resource availability information will be the most current. On the other hand, if the update frequency is too low, fewer messages will be needed, but at the risk that GRM will perform application scheduling based on outdated resource availability information.

In addition, other factors may affect the update frequency. For example, if an LRM is running on a mobile device, like a PDA or laptop, one such factor would be the energy level of the battery, since wireless network communication is known to consume significant energy. Thus, when running at low energy levels, it would be convenient to reduce the update frequency in an effort to help extend battery life.

We have successfully used the dynamic interceptor model to modify the update frequency in a transparent and dynamic way. The level of energy of the battery is the information used by the *Monitor* to decide when the appropriate interceptor must be installed and activated. Nevertheless, as the LRM implementation has not yet been ported to small-form-factor mobile devices, test were only carried out on more capable laptop computers, leaving open the question as to whether this feature will have a significant impact on small devices.

5. Related Work

AutoGrid [Sallem et al. 2007] has similar objectives to our dynamic interceptors model, as both deal with the introduction of dynamic adaptation mechanisms on InteGrade. However, there are differences in the way they enable InteGrade to deal with the dynamism of opportunistic grids.

AutoGrid intends to create an autonomic grid middleware from InteGrade through the addition of new components in its infrastructure. For that reason, the InteGrade architecture has been modified to accommodate these new components. The model of dynamic interceptors, on the other hand, only modifies the underlying ORB OiL. In this way, there was no need to change the InteGrade architecture itself. AutoGrid, however, deals with the dynamic adaptation of structural (as opposed to behavioral) features, such as the scheduling algorithms employed by GRM. Thus, it is reasonable to consider both works as complementary solutions to achieve a high degree of dynamism on InteGrade.

We argue, however, that structural (architectural) adaptation mechanisms provided by an underlying component-based communications middleware, such as Meta-

ORB [Costa and Santos 2004] could also be used with similar effects, thus preventing adaptation mechanisms to be re-implemented at the grid middleware level. This is a promising subject for future research.

AutoMate [Agarwal et al. 2003] is a framework for autonomic grid middleware based on the OGSA standard [Foster and Kesselman 2004]. It is capable of supporting autonomic applications with dynamic auto-adaptation features. The dynamic interceptors model, on the other hand, has similar goals in the context of InteGrade. As it is implemented in the OiL middleware, thus not changing the programming model of InteGrade, there is no need to create grid applications by following a standard defined by some framework, which is the approach followed in AutoGrid. The major difference between the two works is the implementation approach. AutoMate is implemented as a layer between the grid middleware and the application. Our model of dynamic interceptors, in contrast, is implemented on the layer below the grid middleware, more specifically in the communications middleware, thus providing greater transparency for the applications programmer.

6. Conclusion and Future Work

In this paper, we presented a model of dynamic interceptors used to provide runtime adaptation capabilities to the InteGrade grid middleware. This is achieved by introducing behavioral reflection, through interceptors, at the level of the communications middleware that enables interaction among InteGrade's components.

The dynamic interceptors introduced in OiL enabled this ORB to manipulate its non-functional properties according to variations in the execution environment. Using this dynamic adaptation feature, InteGrade can optimize its behavior in a dynamic way.

However, only the InteGrade components that use OiL for communication, such as LRM and GRM, can make direct use of such feature. Thus, for InteGrade to take full advantage of dynamic interceptors, this adaptation mechanism must also be present in JacORB, which is used by the components written in Java.

To demonstrate the benefits of the model, two non-functional properties were implemented. The first one consists of using interceptors to transparently redirect the connections made by LRMs, which, instead of being made directly to the *Cluster Manager* or to other LRMs, are directed to an application proxy through interceptors. Through this proxy, LRMs can communicate with the *Cluster Manager* or other LRMs, even when these nodes are on administrative domains that are protected by firewall and NAT.

The second non-functional property implemented in this work consists of modifying the frequency of update messages from the LRM to the *Cluster Manager*, based on the energy level of the battery.

From these experiments, we have shown that static grid middleware, such as InteGrade, can be made to provide dynamic adaptation features without the need to add new components and mechanisms to the architecture or even to modify the existing components. The only change that was needed was the introduction of dynamic interceptors in the underlying communications middleware. We acknowledge that an extra degree of adaptability, mainly related to the structure of the middleware, will require further adaptation mechanisms. However, for simpler behavioral adaptations, the interceptor approach has significant advantages in terms of transparency.

For the next stages of this work, we intend to implement further non-functional properties that enable InteGrade to adapt with respect to other dynamic aspects of the grids, such as security and fault tolerance. Another goal consists in the implementation of the dynamic interceptors model in JacORB. In this way, the components that use this ORB (i.e., those written in Java) will also be able to take advantage of the dynamic adaptation features. Finally, we aim to apply the same approach to structural adaptation of the grid middleware, by using architectural reflection mechanisms provided by a newer, componentized version of OiL [Nogara 2006].

Acknowledgements

This work has been partly supported by CNPq-Brazil (grant number 55.0895/2007-8) and by FAPEG (Call 02/2007).

References

Agarwal, M., Bhat, V., Liu, H., Matossian, V., Putty, V., Schmidt, C., Zhang, G., Zhen, L., and Parashar, M. (2003). Automate: Enabling autonomic grid applications. In *In the Autonomic Computing Workshop, 5th Annual International Active Middleware Services Workshop (AMS2003*.

Costa, F. and Santos, B. (2004). Structuring reflective middleware using meta-information management: The Meta-ORB approach and prototypes. *Journal of the Brazilian Computer Society*, 10(1):43–58.

Foster, I. and Kesselman, C. (2004). *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, San Francisco, CA, USA.

Goldchleger, A., Kon, F., Goldman, A., Finger, M., and Bezerra, G. C. (2004). InteGrade: Object-Oriented Grid Middleware Leveraging Idle Computing Power of Desktop Machines. *Concurrency and Computation: Practice and Experience*, 16(5):449–459.

Ierusalimschy, R. (2006). *Programming in Lua, Second Edition*. Lua.Org.

Maia, R., Cerqueira, R., and Kon, F. (2005). A Middleware for Experimentation on Dynamic Adaptation. *ARM'05*.

Nogara, L. G. C. (2006). *Um Estudo Sobre Middlewares Adaptáveis*. Master thesis, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, Brazil.

OMG (2004). CORBA Firewall Traversal Specification. http://www.omg.org/docs/ptc/04-03-01.pdf, accessed: January 2008.

OMG (2008). *Common Object Request Broker Architecture: Core Specification, Version 3.1*. Object Management Group.

Sallem, M. A. S., de Sousa, S. A., and da Silva e Silva, F. J. (2007). AutoGrid: Towards an Autonomic Grid Middleware. In *Enabling Technologies: Infrastructure for Collaborative Enterprises - WETICE 2007*, pages 223–228.

Schmidt, D., Stal, M., Rohnert, H., and Buschmann, F. (2000). *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects Volume 2*. John Wiley & Sons, Ltd, New York, NY, USA.

Spiegel, A. (2005). JacORB - The free Java implementation of the OMG CORBA Standard. http://www.jacorb.org.