A Service-Oriented WSRF-based Architecture for Metascheduling on Computational Grids ⋆

G. Moltó *, V. Hernández, J.M. Alonso

Departamento de Sistemas Informáticos y Computación. Universidad Politécnica de Valencia. Camino de Vera S/N, 46022 Valencia, Spain

Abstract

This paper describes a Grid service developed over the Globus Toolkit 4, which provides multi-user resource brokering on computational Grids. Both the architecture and implementation details are covered, emphasizing the usage of the WSRF specifications and GSI-based security to create a generic, secure and interoperable metascheduler. In addition, we also describe the development of a graphical client-side application that provides an ubiquitous access to the metascheduler service. This work is currently being employed in production for the execution of a biomedical application that simulates the electrical activity of cardiac tissues.

Key words: Grid computing, Metascheduling, Globus Toolkit, WSRF, Service-Oriented Architectures

1. Introduction

Grid Computing technologies [1] have emerged as a solution for the computational problems of Virtual Organisations (VOs), enabling the collaborative usage of remote resources to satisfy the computational requirements of time-consuming tasks.

Among all the available Grid middlewares, the Globus Toolkit (GT) [2,3] is broadly accepted to be the current de facto standard for deploying computational Grids. However, GT only provides the basic services and capabilities to deploy and use Grid infrastructures. Performing remote executions of scientific applications on a computational Grid typically requires the usage of *metascheduling* technologies [5], that provide all the functionality required for task management.

Metascheduling is the process that efficiently allocates a set of tasks on the available computational resources of the different organisations that compose a Grid infrastructure. We envisage Grid metascheduling technologies as a set of interoperable components which abstract all the underlying complexity of the Grid middleware, which offers the execution support. Only if these Grid technologies get closer to the end user, they will be adopted in many scientific fields.

In this paper, we go one step beyond the traditional usage of metascheduling and we propose a metascheduler Grid service that can be accessed throughout the network by users interested in task allocation on computational Grids. It has been developed by exposing a Grid service interface to GMarte [6,7], a previously developed metascheduler.

With the proposed approach, the user just focuses on defining the computational tasks to be executed, and delegates to this Grid service their efficient execution on the available Grid infrastructure. As opposed to traditional metaschedulers, which typically require an installation and configuration on the client machine, and are typically bounded to a given platform (usually Unix-based), we propose a Service-Oriented Architecture (SOA) that can be used from different architectures and operating systems. This approach enables to dramatically simplify the usage of computational Grids for job executions, even accessible for Windows-based desktop PCs, thus paving the way for the widespread adoption of Grid technologies by the least experienced users.

In addition, creating a generic and interoperable component based on standard technologies contributes to the ecosystem of tools employed to interact with computational Grids.

The remainder of the paper is structured as follows. First, section 2 introduces the metascheduling framework and the abstraction environment provided by GMarte, which is the

⋆ The authors wish to thank the financial support received from the Spanish Ministry of Science and Technology to develop the project GRID-IT (TIC2003-01318). This work has been partially supported by the Structural Funds of the European Regional Development Fund (ERDF).

* Corresponding author: Tel. +34963877356. Fax.: +3496877359
Email address: gmolto@dsic.upv.es (G. Moltó).

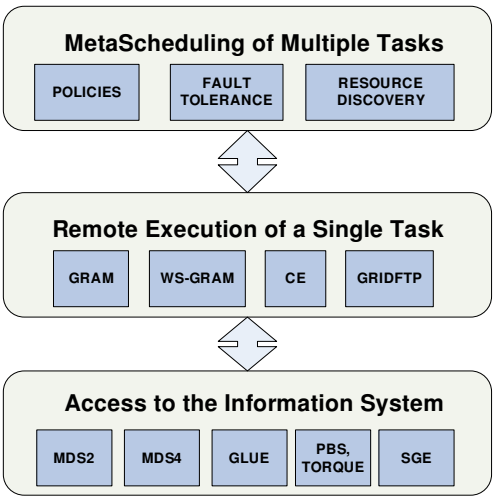


Fig. 1. Layered diagram with the abstraction framework provided by GMarte.

starting point of this work. Then, section 3 explains the architecture and functionality of the GMarte Grid Service as well as the development of a graphical client to provide an easy interaction. Later, section 4 describes the utilization of the Grid service to offer transparent execution services to a biomedical research group that simulates the cardiac electrical activity. Section 5 compares the approach suggested in this paper with others found in the literature and finally, section 6 summarises the paper.

2. The GMarte Framework

GMarte [6,7] is a software framework, developed by our research group, aimed at simplifying the process of executing scientific applications on computational Grids based on the standard Globus Toolkit [2,3]. Behind this simple description, there is a tremendous effort to enable the users to rapidly deploy their applications for execution on a computational Grid. GMarte has been developed as a Java library which exposes a high level API (Application Programming Interface) to the user. The GMarte framework is a layered software (see Figure 1) that first introduces an homogeneous interface to access the Information Systems of the heterogeneous computational resources. Over this common API, a layer which enables remote task execution on Globus-based resources was implemented. Later, we extended the capabilities of GMarte by to introduce metascheduling functionality for the allocation of multiple tasks on computational Grids.

Therefore, the GMarte framework is not only a metascheduler but an abstraction environment that enables to interact with computational Grids at several levels. In the following subsections, we overview the fundamental aspects of GMarte concerning each layer mentioned.

2.1. *Information System Management for Computational Resources*

The computational resources belonging to a Globus-based deployment expose very valuable information commonly employed for activities related with the metascheduling process. For example, in order to decide the most suitable machine for the execution of a given task, a *resource selection* must be performed. This procedure obtains the *dynamic information* of each machine, gathering the available number of processors, the available amount of RAM, etc.

However, this information is exposed in a different manner, depending both on the Globus Toolkit version and the LRMS (Local Resource Management System), that is, the job manager, if any, on the remote machine. For example, in the Globus Toolkit 2.4.3, the MDS2 (Monitoring and Discovery Service) offers the computational information through a server typically listening in the port 2135, accessible via LDAP (Lightweight Directory Access Protocol) interfaces. However, the Globus Toolkit 4 exposes the computational information via the *DefaultIndexService* (MDS4), which is just one of the multiple services deployed in the Grid services *container*. In this case, the information is published in XML (eXtensible Markup Language) format and typically accessed via XPath (XML Path Language) queries.

In order to hide all this complexity of protocols and data formats from the end user, GMarte provides an abstraction layer to access the computational information. For that, we introduced a high level object, called *GridResource*, which offers a common API with user-level methods (getAvailableProcessors, getAvailableRAM, etc.) that gather the requested information, in a transparent way for the user. This is achieved by using the appropriate protocols depending on the underlying Grid middleware of the remote machine.

This feature is of paramount importance for the end user, which no longer has to deal with the low-level implementation details and data formats to access the computational information. Additionally, GMarte implements a transparent caching component that distinguishes between *static* information, which it is assumed to remain unchanged (i.e., operating system name, CPU architecture, etc.) and *dynamic* information (i.e, available number of processors, available RAM, CPU load, etc.). The queries to static information are automatically cached so that subsequent ones do not produce access to the remote information system.

At the moment, the features described within GMarte are available for resources with the Globus Toolkit 2.X and 4.X, as well as the Computing Elements in LCG-2 [4] deployments. The supported LRMS are PBS/Torque (Portable Batch Systems), and SGE (Sun Grid Engine).

2.2. *Abstracting the Process of Remote Task Execution*

GMarte enables to abstract the process of remote task execution by performing fault-tolerant executions with coordinated data staging, even for parallel applications using the MPI libraries.

First of all, the application dependent input files are transferred via GridFTP from the client to the remote machine (a process called *stage-in*). Then, the application is started in the remote machine via the GRAM (or WS-GRAM) service provided by the Globus Toolkit and periodically monitored to detect failures. When the execution finishes, the output files generated are transferred to the client machine via GridFTP (a phase called *stage-out*).

In this layer, GMarte implements a multi-level fault tolerance scheme which allows to recover itself from failures both during the data transfers (which are retried a certain amount of times before giving up), as well as during task execution, where failed tasks are marked so that the metascheduler decides the appropriate action to be taken. Failures in remote computational resources cause a re-scheduling of the tasks on other available machines.

In addition, GMarte supports the definition of application-dependent checkpoint files which are periodically transferred to the metascheduler machine so that failures in the tasks or resources are handled to retake the execution from the latest checkpoint. For long-lasting execution this can save a considerable amount of time.

2.3. *Metascheduling Approach*

In brief, the metascheduling procedure typically involves several phases [5] in order to efficiently solve the problem of executing a set of tasks on the available remote resources (see Figure 2). In GMarte, these phases are handled as follows:

First of all, the *Resource Discovery* phase involves the discovery of a set of candidate execution machines from either a GIIS (Grid Index Information Service) or a BDII (Berkeley Database Information Index). These components aggregate resource information and are queried to provide a list of machines that satisfy some computational features specified by the user.

Once a list of potential execution machines have been obtained, the *Resource Filtering* phase is in charge of discarding those resources that either are not accessible with the user credentials or do not satisfy the computational requirements of the task to be executed.

When both previous phases finish, a set of computational resources on which to perform the execution of the tasks is available. In order to decide which resource is going to run each task, a *Resource Selection* stage must take place for each one, which chooses the current best machine on which to execute it, according to some criteria. GMarte implements different resource selection policies. The most commonly used involves a performance model that tries to minimise the execution time of the application, considering both the number of available processors (for parallel executions) and the data transfer cost between the submission machine and the computational resource, which is dynamically updated upon each data transfer. In addition, being an abstraction framework, the user can easily create new policies by providing different score functions for computational resources. As no resource selection can be considered the best for all kind of applications, this allows to extend the functionality via additional, user-programmed modules. This procedure involves creating a new Java class which ranks the resources according to the new criteria. By using the high level API to access the computational information of resources, this can be a simple task.

GMarte implements a multi-threaded metascheduler that enables to concurrently perform the distinct phases involved in the remote task execution of different tasks (resource selection, stage-in, execution and stage-out). For example, the user can specify the number of threads involved in the stage-in phase. If 3 threads are for example specified, then up to 3 tasks will be simultaneously handled.

This approach enables to introduce a considerable speed-up in the whole task allocation process, but specially in the start-up time, where the resource selection phase typically requires a time linear with the number of computational resources. If a large testbed is used, this selection phase may become a time-consuming process due to the retrieval of dynamic information of each machine. Using multiple threads for resource selection, combined with a client-side processor reservation stage to avoid selecting the same resource more than once, offers important improvements in speed compared with a single-threaded approach.

Finally, it should be mentioned that the functionality described can be used via a GMarte Java application as well as through XML documents, which are automatically mapped to the corresponding abstraction objects. Being based on Java, GMarte can be seamlessly run both on Windows and Unix machines.

Currently the GMarte framework has been successfully applied for the execution of a biomedical application that simulates cardiac electrical activity both on local and large-scale Grid infrastructures [8]. Its functionality has also been introduced as part of a Grid service that performs the structural analysis of buildings [9]. This has enabled to perform executions of scientific applications in a transparent manner for the user.

3. GMarteGS: A Grid Service for Metascheduling

In order to create an interoperable metascheduler, we decided to migrate the GMarte framework to a SOA based on standard technologies provided by Grid services.

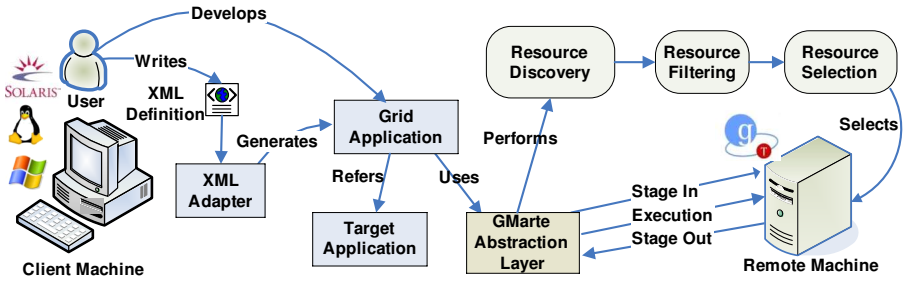


Fig. 2. Overview of the principal phases involved in the GMarte framework.

3.1. *Grid Services, OGSA and WSRF: Globus Toolkit 4*

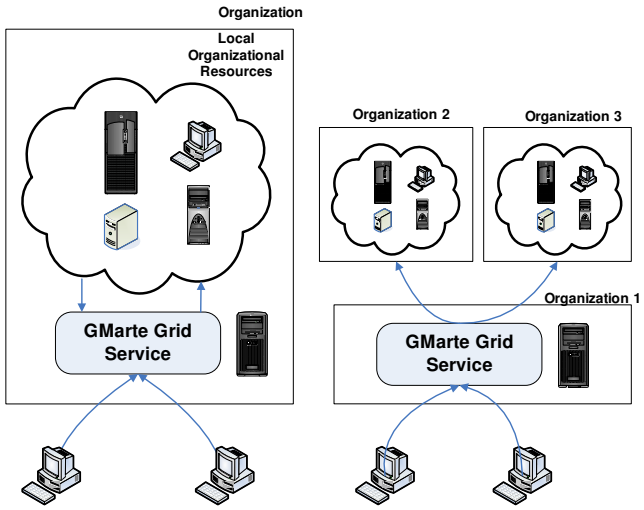
The evolution of the Grid Computing field indicates that there is an unquestionable trend in producing loosely-coupled, interoperable and generic components.

The Globus Toolkit (GT) has been developed since the late 1990s to support service-oriented distributed computing applications and infrastructures [2]. Initially, the early versions of GT were based on services that interacted among them using proprietary interfaces, although commonly implemented using standard protocols like LDAP or FTP (File Transfer Protocol). With the release of the GT version 3 (GT3), a step towards Web services was performed, trying to expose common and standard interfaces for the interaction with the different services. GT version 4 (GT4) represented an important advance in terms of functionality, conformance to standards and usability [3].

GT4 provides the implementation of a set of Grid services which conform to OGSA (Open Grid services Architecture) [10]. OGSA represents an evolution towards a Grid system architecture based on the concepts and technologies provided by Web services. Being developed by the Open Grid Forum (OGF), OGSA defines a common, standard and open architecture for all the services that can be found in a Grid system (job management, resource management, security, etc) [11], so that tools of different vendors can cooperate together using standard interfaces. As a result, OGSA defines its underlying architecture to be based on special Web services, which maintain their state from one invocation to another, something that traditional Web services are not able to. This is where WSRF (Web Services Resource Framework) [12] comes into play, specifying how Web services can be stateful. This is achieved by coupling a data container (WS-Resource), which stores the stateful data, to a web service. OGSA uses this new concept to specify the underlying architecture of the Grid services. Therefore, GT4 includes an implementation of WSRF as well as a set of Grid services developed on top of WSRF, which are compliant with OGSA requirements.

Migrating GMarte to a SOA means moving the resource brokering functionality from the client machine to a specialised machine which now performs the task allocation functionality for multiple clients by means of a Grid service.

This represents a two-fold strategy. On the one hand, lighter clients can be developed since they no longer carry out the metascheduling process. This implies decoupling the task allocation (run in the Grid service) with the state visualisation (done in the client), thus conforming to a loosely-coupled architecture.



(a) Enterprise Grid (b) Global Grid

Fig. 3. Different scenarios on which the GMarte Grid Service approach can be applied.

On the other hand, this scenario opens new possibilities such as those depicted in Figure 3. The GMarte Grid Service (GMarteGS) can be deployed within an organisation as an entry point to its local computational resources, thus creating an Enterprise Grid (Figure 3.a). This enables an organisation to provide secure and trusted computational access to multiple clients willing to execute their tasks in the organisation resources. Alternatively, GMarteGS can be deployed within an organisation to enable performing remote task executions on computational resources from other institutions (Figure 3.b). This way, the Grid service would represent a gateway among different machines and the clients that want to access distributed computational infrastructure.

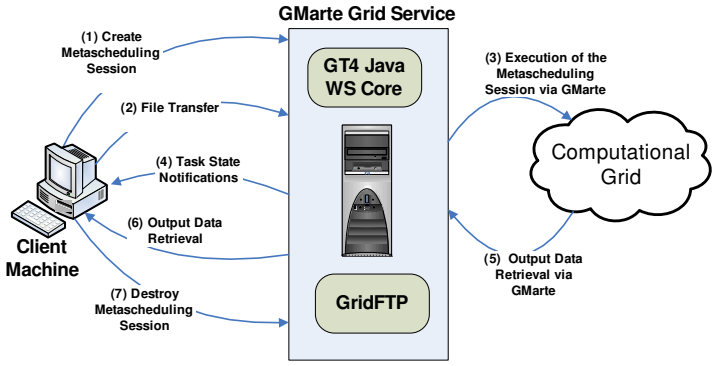


Fig. 4. General functionality provided by the metascheduling Grid service.

3.2. *Main Functionality and Interaction*

The aim of GMarteGS is to provide multi-user transparent metascheduling functionality to perform executions on computational Grids. Therefore, we introduce the concept of a *metascheduling session* and the service provides different operations to manage them. A metascheduling session is defined by the computational tasks to be executed. Additionally, the user can also specify the computational resources to be employed and the configuration of the metascheduler. This way the user can select the Grid infrastructure or let the Grid service decide.

All the information exchanged between the client and the service is performed in XML language. The main reason is that XML enables to structure information in a manner that is both easily understood by humans and machines. In addition, being a text-based information can be obtained as the result of a method invocation, thus easing communication between the client and the service.

The principal interface of the Grid service covers a set of methods that allow the client to:

(i) Create a Session. Creates a new metascheduling session.

(ii) Initialise the Session. This operation enables to specify the XML documents (in the shape of a String as method arguments) that describe the tasks and, optionally, the computational resources and the configuration of the metascheduler. The user may also decide to rely on the resource discovery functionality to gather a list of computational resources from an Index Service. In addition, user credentials are delegated to the service so that authentication against computational resources is performed by the Grid service on behalf of the user. Notice that it would also be possible to let the Grid service use some generic credentials to interact with the computational Grid, regardless of those provided by the user. This can provide a computational gateway between different Grids accessed with different credentials.

(iii) Gather Session Information. Retrieves information about the session, such as its name, the data folder in the Grid service machine that will host the client data, the time spent on the execution of the session, etc.

(iv) Gather Session State. This method obtains the current state of the metascheduling session. This document specifies, for each task, the machine hosting the execution, the number of processors employed in case of a parallel execution, and its state according to the life cycle of tasks.

(v) Gather Testbed State. This method obtains the current state of the testbed. It basically details some computational information about the machines being employed, such as the number of available processors. This allows the user to have an overview of the state of the Grid infrastructure.

(vi) Destroy the Metascheduling Session. It instructs the Grid service to stop the task allocation process, thus cancelling all the on-going simulations, as well as erasing any data related to the metascheduling session.

Figure 4 summarises the main interaction between a client and the Grid service. First of all, the client creates a new metascheduling session and specifies the XML definition of tasks. Then, all the data required for the execution of the tasks is transferred via GridFTP to the Grid service machine. Next, GMarteGS performs the execution of the session delegating on the functionality implemented by GMarte. At the same time, the user is notified about changes in the state of the tasks. Once the session has finished, the user can retrieve the output data to the client machine. Notice that it is also possible to retrieve the output results of individual tasks as long as they have finished.

3.3. *GMarteGS Architecture and Implementation*

The principal architecture of the Grid service developed is shown in Figure 5. The Grid service has been split into a *Factory Service* (FactoryGMarteGS) and an *Instance Service* (GMarteGS), following recommendations of [11]. On the one hand, the Factory service is in charge of creating new WS-Resources, which store the stateful data, that is, a *MetaschedulingSession* object. Notice that this creation is actually performed by the *ResourceHome*, which stores and manages WS-Resources. On the other hand, the In-

stance service (GMarteGS) represents the core of the Grid service, and it implements the operations that affect a given metascheduling session. Although it is not necessary to split the functionality in two different services, this approach enables to introduce new scenarios (not covered in this paper) such as having a distributed federation of metascheduling Grid services, where the Factory service diverts the client requests of creating new sessions to the most appropriate metascheduler Grid service, based on availability, priority or any other parameter.

The interaction between the client machine and the metascheduler Grid service is also covered in Figure 5. First of all, the client needs to know the URI (Uniform Resource Identifier) of the FactoryGMarteGS Grid service, which represents a unique address to identify this service.

Next, a new metascheduling session is requested to be created (step 1). This causes (step 2) the ResourceHome to create a new WS-Resource (step 3), which contains the MetaschedulingSession object that hosts the session information, returning a unique identifier (Resource Key) of the newly created WS-Resource (step 4). Using this Resource Key plus the URI of the GMarteGS service, an endpoint reference (EPR) is created and returned to the client (step 5, 6). An EPR, in this context, unambiguously identifies both the GMarteGS service and the metascheduling session (WS-Resource) that will be affected by the operations invoked.

Later, the client invokes any of the methods published by GMarteGS (step 7). Then, using the ResourceHome this service can automatically locate the WS-Resource the method will work with, from the EPR (step 8) and the operation is performed (step 10), only affecting the correspondent MetaschedulingSession object. Finally, the result of the method invocation is returned to the client (step 11).

3.4. *WSRF-based Specifications*

GMarteGS uses several WSRF specifications given that the adoption of standards enables to create interoperable components.

First of all, the WS-Resource specification describes the relation between a service and a resource as well as the access mechanisms to resources through interfaces based on Web services. In GMarteGS, each metascheduling session corresponds to a WS-Resource. This enables to isolate sessions of different users. The fact that each WS-Resource has a unique identifier enables to automatically know the session on which a certain operation has to be invoked, given an EPR.

Secondly, the WS-ResourceProperties specification standardizes the mechanism to define the properties of a resource, which represent their state and how they can be declared as part of the service interface. In GMarteGS, each WS-Resource has a resource property represented by the XML summary of the state of the metascheduling session, thus capturing the state of the WS-Resource.

The WS-ResourceLifeTime specification defines mechanisms to destroy a WS-Resource. Basically, it specifies two different strategies: immediate and lease-based destruction. GMarteGS enables to destroy a metascheduling session at any moment cancelling the execution of its tasks. In addition, it implements an automatic mechanism, based on this specification, to destroy finished sessions that have been idle (no operation has been performed with the session) after 15 days. This enables clients to use the Grid service machine also as a temporary data repository for the generated output data from executions.

Finally, we have used some WS-Notifications specifications, particularly WS-Topics and WS-BaseNotification, although they are not part of WSRF but available in GT4. They allow part of a system to be notified when an event occurs in another point of it. Thus, a publish-subscription method based on topics is introduced where a client subscribes to a topic in order to receive notifications related to it. This feature has been incorporated to GMarteGS so that changes in the state of the tasks are automatically notified to the client by means of XML data describing the current state of the tasks. As initial and final phases of metascheduling cause many task state changes we have implemented basic contention mechanisms to aggregate multiple notifications that occur within the same time interval in order to reduce the communication overhead.

The usage of a notification mechanism reduces the workload in the service as the multiple users no longer require a periodical query to track the state of the tasks. Only when changes are detected a communication is performed. However, the usage of notifications relies on the installation of at least the Java WS Core of GT4, in the client machine, to be able to receive the notifications. This is a handicap when creating lightweight components to interact with the Grid service.

3.5. *Security Infrastructure*

Security in a multi-user environment accessible from the Internet is a crucial aspect. Therefore, it is important that the service guarantees the principal features of a secure communication: privacy, integrity, authentication and authorization [13]. To address this problems we have used standard mechanisms based on GSI (Grid Security Infrastructure) provided by the Globus Toolkit 4.

Figure 6 shows the interaction diagram between the client and the service detailing the security aspects involved. The client must have available a user certificate signed by a Certification Authority (CA) on which GMarteGS trusts. Otherwise, it is impossible to perform a communication.

The FactoryGMarteGS service has been configured to require GSI Secure Conversation to establish a dialog. This mechanism forces to automatically perform a mutual authentication so that both the client and the service ensure that they are talking to the expected partner. In addition,

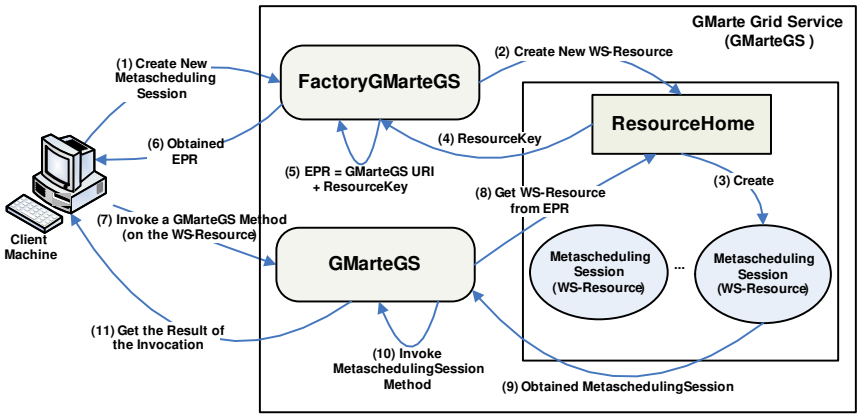


Fig. 5. Architecture of the metascheduler Grid service as well as client-service information flow.

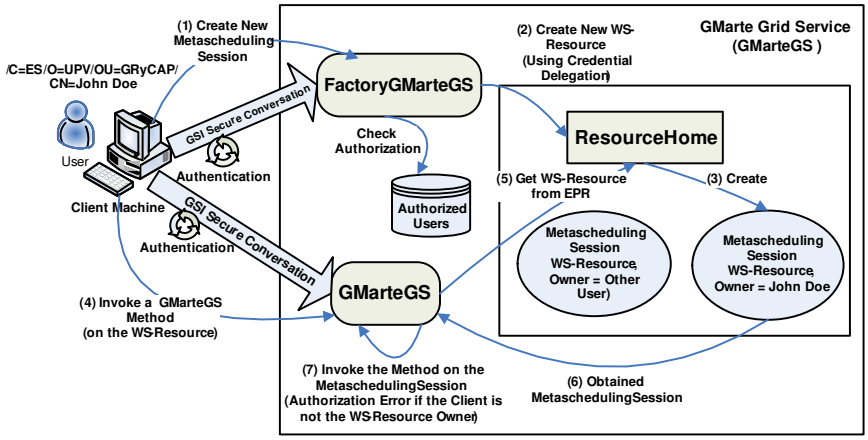


Fig. 6. Diagram of the client-service interaction with the security mechanisms employed.

the channel is configured to support privacy and data integrity. Once the connection has been established, the FactoryGMarteGS service queries a database of users, thus granting access only to authorized ones.

The current security scheme forces the client to use a credential delegation mechanism that enables the service to create a new WS-Resource (metascheduling session) setting its owner to the user credentials. This feature is of fundamental importance as all the accesses to a WS-Resource can only be performed by the creator of the session. This security approach is more secure than a simple access control to authorized users, as access policies to WS-Resources are controlled by themselves.

The security mechanisms implemented achieve the following goals:

(i) All the communications between the client and the service are private and can not be altered by a third party without the receiver to notice it.

(ii) An user which has not been registered as an authorized user of the Grid service can not create metascheduling sessions.

(iii) Only the creator and owner of a session can invoke an operation on it. Any other user authorized by the system has no access to this session.

3.6. *The GMarteGS Client GUI*

GMarteGS uses standard protocols like SOAP (Simple Object Access Protocol) and XML for data exchange, and WSDL (Web Services Definition Language) for the interface specification. Therefore, a client using such protocols could programmatically interact the service, no matter the programming language employed. Additionally, a client library implemented in Java has been developed for that purpose. First of all, this library implements methods for data management between the client and the service. It enables to automatically perform data transfers from the XML description of tasks provided by the user. In addition, it exposes high level methods to interact with GMarteGS. The development of this client library enables to incorporate the service functionality within applications that, using the API provided, can delegate the execution of tasks to the Grid service.

Alternatively, we have developed a commodity graphical client that provides easy-to-use interaction with the service for those users that just want to perform remote task execution but do not require a full API (see Figure. 7). This application allows to:

(i) Perform data transfers between the client and the service, both for the input and output data of the

tasks.

(ii) Specify the different XML documents that are required to create a new metascheduling session. It is mandatory to provide at least the definition of tasks.

(iii) Show the current state of the metascheduling procedure by means of populating tables that summarise the process.

(iv) Know the current state of the testbed, describing the main features of the computational resources involved.

To enable an easy access to GMarteGS, we have relied on the Java Web Start technology (JWS) to create a self-contained client that can be accessed by only means of a Java-enabled web browser. This feature provides ubiquitous access functionality to our client, which can be accessed from every platform that supports Java.

JWS enables to deploy a full Java application to a web server. This technology provides all the support to automatically retrieve the latest version of the application and starts the application in a sandbox in the client machine.The usage of a cache mechanism enables to save bandwidth from the client-side, while ensuring that the latest version of the application will always be deployed to the client machine.

Before working with the application, the user must have available in the client machine all the components required for security based on GSI. This information typically involves:

(i) An user certificate signed by a CA, which certifies that the public key included in the certificate belongs to the user. This way, the user can be recognised as such by the computational resources in the Grid that trust this CA.

(ii) A private key, only readable by the user, which enables to decipher messages addressed to the user which were ciphered with his/her public key. In GSI security, the private key is also protected by a password to introduce an additional level of security.

(iii) A certificate of the CA that signed the host certificate of the Grid service machine.

The security configuration on the client machine is a one-time process, provided that any certificate involved does not expire. In order to ease this procedure, we have also deployed under JWS the configuration setup of the Java Commodity Grid Kit [14], a step-by-step GUI that enables to configure the client machine for GSI security. In addition, proper CA certificates, which signed the computational resources certificates, must reside in the Grid service machine to access the machines during the task allocation procedure.

Once the GSI configuration has been performed, the client will always interact with the computational resources through the Grid service, via an user *proxy*, which implements a single sign-on strategy that prevents the user from typing the password each time the private key is accessed. The proxy is just a small file that holds the user credential for a limited amount of time.

After the security configuration procedure, the user can now access the Web site that hosts the GMarteGS Client GUI component. This process automatically starts in the client machine the JWS support to download the requested component. This involves retrieving the GMarteGS Client GUI software itself and all its dependent libraries (a set of JAR files), which broadly cover the following areas:

(i) GMarte libraries employed for reliable data transfers, accessing XML documents, etc.

(ii) Cryptographic libraries required for the GSI-enabled security infrastructure.

(iii) Logging components that trace the execution of GMarte and other Java modules.

(iv) Support libraries to access the WSRF-based Grid services, which includes components such as AXIS modules for handling Web services and WSDL modules for the description of the service interface.

All this information represents a total 8 MBytes to be downloaded to the client machine. Considering the cache that implements JWS, a lightweight access to the client GUI is provided.

Before the application is started in the client machine, the user is notified that the software demands full access to his/her machine (in order to read certificates, perform GridFTP data transfers, create directories, etc.). In addition, the certificate information that signed the application is shown. Upon acceptance, the GUI component is launched for the user to interact with the Grid service.

To ensure client multiplatform, successful tests have been performed in several architectures (Pentium IV, Intel Xeon, AMD Opteron and UltraSPARC-III) and operating systems (Linux Fedora Core, Windows XP and Solaris 10).

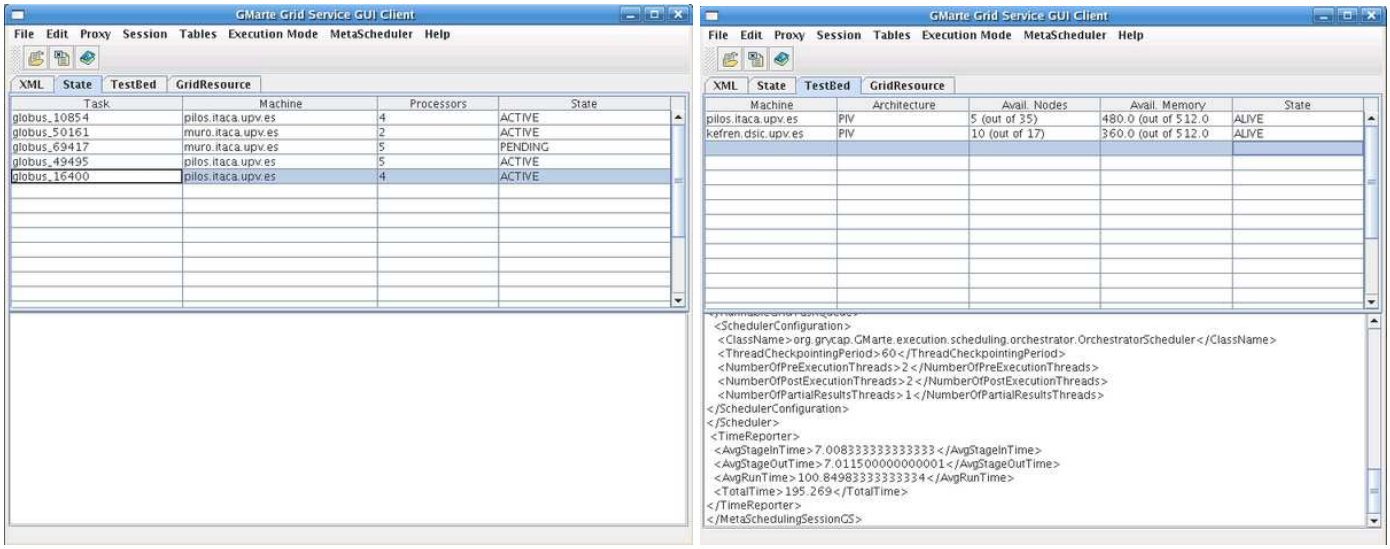
3.7. *Fault-Tolerance*

Using a service-oriented approach where client and server reside in different machines introduces new problems which must be managed.

First of all, failures in the client should not affect the executions being carried out in the remote computational resources. The strategy implemented in GMarteGS relies on the fact that an EPR uniquely identifies both the Grid service and the WS-Resource that represents the metascheduling session.

Therefore, when a new metascheduling session is created, its EPR is automatically stored. Therefore, a failure in the client can be managed by using a new one that, using the stored EPR, accesses the session. This functionality enables to decouple the client and the service so that a user can create a session, submit the executions and switch off the computer. Later, the client will be able to reconnect to the session to track the state of the metascheduling procedure.

Failures occurred in both the computational resources or the tasks are managed by the GMarte metascheduler, which already provided this functionality. Therefore, the service is only aware of changes in the state of the tasks.



(a) (b)

Fig. 7. Snapshots of the graphical client developed to interact with GMarteGS.

In order to handle failures in the service itself, we have implemented a persistent approach. By default, WS-Resources reside in the main memory, so a failure in GT4 container or in the service causes those resources to disappear. Therefore, we have employed the Java serialization mechanism to store a copy of the metascheduling session in secondary storage. In the case of failure, it is possible to restart the GT4 container and any operation performed in a session stored in disk will cause its automatic loading to principal memory. This enables to retake the metascheduling session for those tasks that still had not finished.

4. An Application of the GMarte Grid Service: Computational Gateway

Nowadays, GMarteGS is being used to provide computational access to the Grid resources located at our research group for the study of cardiac pathologies. The understanding of cardiac electrical activity is fundamental for the development of new therapies to address diseases such as ventricular fibrillation, the most mortal of arrhythmias. For several years now our research group has a collaboration with the Center for Research and Innovation on Bioengineering (Ci^2B), at the Universidad Politécnica de Valencia. We have developed a High Performance Computing-based cardiac simulator which enables to reduce the execution time of cardiac simulations in a cluster of PCs [15]. This is an MPI-based application which solves the reaction-diffusion problem of action potential propagation on cardiac tissues. The simulator employs computationally intensive cardiac cellular models and requires the resolution of a large sparse linear equation system in each simulation time step. The usage of MPI-2 I/O enables to collaboratively store on disk the large volume of biomedical data by the different processors in a parallel execution.

However, there are many cardiac studies that require to perform multiple simulations. For example, studies of vulnerable window in ischemia need to vary the time interval between two consecutive electrical stimulus in order to detect the range of values which provoke a reentry, a phenomenon that can derive into heart fibrillation. Besides, to evaluate the influence of certain medicines, it is necessary to modify the concentration of drugs to study how they affect the electrical activity of the tissue. These cardiac case studies are composed of multiple independent simulations which are ideal for their execution on a Grid infrastructure.

As part of this collaboration, GMarteGS has been deployed as the entry point to a Grid infrastructure composed by several clusters of PCs within our research group. The Client GUI application enables the experts to focus on the definition of the cardiac case studies, while GMarteGS provides a transparent support for the parallel execution of the different simulations on the machines that compose the Grid infrastructure. As a result, biomedical experts can easily enlarge their computational power, using a Grid infrastructure.

Figure 8 summarises the interaction with the Grid service by the biomedical experts. Notice that many clients can concurrently work with the service. In this particular situation, the Client GUI application has been restricted so that users need only specify the tasks to be executed.

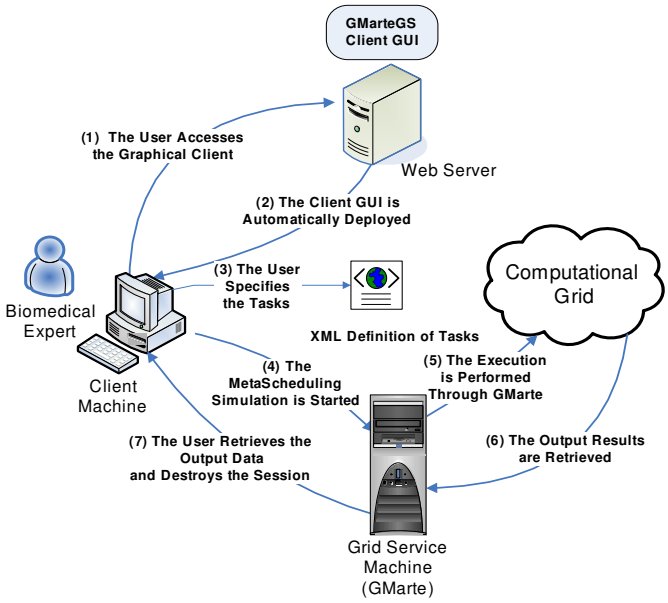


Fig. 8. Usage of GMarteGS to provide biomedical experts with transparent access to a computing infrastructure.

4.1. *A Computational Case Study: Analysis of Ischemia on the EGEE Testbed*

The implemented Grid service supports the capability to delegate executions on the EGEE testbed [17], the largest production Grid infrastructure all over the world. This functionality enables to transparently access hundreds of computers, via the high level interfaces in GMarte, for the execution of computationally intensive applications.

In order to assess the benefits of using a large Grid infrastructure, we have instrumented the execution of a typical cardiac case study on the EGEE testbed, via the SOA developed. The case study analyses the effects of myocardial ischemia, a condition of the tissue caused by oxygen deprivation to the heart muscle, in the electrical propagation. The generation of ventricular tachycardia and fibrillation can be studied using a model of a regionally ischemic tissue [16]. This study introduces the conditions that appear during the first 10 minutes of myocardial ischemia, on a central zone of 50x50 cells in a 250x250 cells bidimensional virtual tissue. Using a time increment of 0.5 minutes, 21 independent parametric executions are required. Each simulation reflects the conditions of the tissue on the specified minutes after the onset of ischemia. Each execution simulates 80 ms and stores a snapshot of the membrane potential of all the cells of the tissue every 0.2 ms in the interval [60,80] ms, generating a total amount of data of 47 MBytes per simulation.

For the execution, we have relied on the resources of the Biomed Virtual Organisation (VO), within the EGEE testbed. As MPI executions are currently not supported by LCG-2, we have disabled parallel executions on this infrastructure. The involved LCG-2 services are the Storage Element (SE), for storing both the input and output data of a simulation, the Resource Broker (RB), for submitting the execution, the Computing Element (CE) and its corresponding Worker Nodes (WN), where the executions take place, the Logging & Bookkeeping (LB), for monitoring the state of the tasks, and the LFC (LHC File Catalog), to manage file replicas on the Grid. All these services are transparently orchestrated, in GMarte, in order to achieve the data management, the delegation of execution and the monitoring of tasks. This VO has currently more than 1800 queues available for execution.

Concerning the execution results, the metascheduling session lasted for 1.71 hours, since the task allocation started until the output data of the last task were retrieved. The resource selection policy of the RB was to choose the machine with the largest number of free CPUs. As a consequence, all the executions were allocated to the same machine, which had 160 available CPUs out of a total 190 WNs. A sequential execution of the whole case study on a Pentium Xeon at 2.8 GHz with 2 GByte of RAM, required 29.4 hours. Therefore, the Grid approach was 17.19 times faster than the sequential approach, enabling to produce more results per time unit. All the executions were concurrently executed in the Grid infrastructure. Therefore, the delays in the Grid approach included the data transfers required between GMarteGS and the SE as well as between the WNs and the SE. It should be mentioned that LCG-2 introduces a substantial delay since a job request is sent until the application gets running on a remote resource. In order to alleviate this problem, a multi-threaded approach is employed, in GMarte, to perform concurrent job submission of different tasks.

Currently, we are planning to use the developed system as a computational gateway to a two-layer Grid infrastructure. The first one would be composed by clusters of PCs from our research group and the second one would be the EGEE testbed. This would enable firstly to use our local resources, which produce shorter execution times (due to the overhead of LCG-2), until they become exhausted. Then, executions would be transparently delegated to an EGEE-based production Grid. This approach would offer a sustained quality of service to the biomedical users. Other examples of execution results on local and regional Grid infrastructures based on the Globus Toolkit, using GMarte, can be found in [8].

Decoupling the scientific activities from the computational infrastructure has allowed biomedical experts to concentrate on the scientific side, while the executions are transparently performed by GMarteGS. Scientists are now able to access a Grid infrastructure with easy-to-use tools that hide the complexity of the Grid. By enabling biomedical experts to enlarge their computational infrastructure, research productivity is easily increased, as more simulations per time unit can now carried out.

5. Related Work

This work addresses the topic of multiplatform metascheduling combined with a WSRF-based approach to create a multi-user metascheduler accessible through standard interfaces and an easy-to-use graphical client.

In the field of resource brokering we find GridWay [18] which is one of the principal metaschedulers available for Globus platforms. GridWay is an open source metascheduler that performs unattended, reliable and efficient execution of jobs on computational resources based both on GT2 and GT4. However, GridWay only runs on UNIX-like operating systems. Moving to a SOA, we find the eNanos Grid Resource Broker [19], a metascheduler implemented in GT3 to manage remote application executions.

To the best of our knowledge, there is little work focusing on metaschedulers exposed via WSRF-based Grid services. In [20], a WSRF-based metascheduler is designed, along with a client application, which performs task allocation to resources. Its Grid service receives single task requests, which cause a resource discovery to occur prior to execution. In GMarteGS, we generalise this approach by introducing the concept of a metascheduling session, enabling whole case studies to be submitted for execution, which only involves one resource discovery per study. In addition, we provide a security infrastructure and emphasize on the usage of high level client-side interfaces which ease the process for the end users.

As another Grid service metascheduler, we can also find the Community Scheduler Framework (CSF) [21], which was originally developed in GT3 and later adapted to GT4. While GMarteGS is able to manage different metascheduling sessions, which can involve different resources, CSF4 receives individual jobs specified in RSL (Resource Specification Language). In addition, CSF4 does not coordinate data staging for the application data. Also, GMarteGS uses standard security mechanism between sessions while no related documentation is found about CSF4. This is the case of the the Gridbus Broker [22], which features a WSRF-based service interface but no security approach appears to have introduced.

6. Conclusions

In this paper, the GMarte metascheduler has been integrated into a Service-Oriented Architecture based on WSRF Grid services, implemented via the Globus Toolkit 4. Using standard functionalities provided by WSRF specifications have enabled to implement a multi-user metascheduler which is generic and interoperable with other services in the Grid ecosystem. In addition, standard security mechanisms based on GSI have been implemented to ensure data privacy and integrity.

The usage of GMarteGS has been described as a computational gateway for the study of the electrical activity of the heart, enabling biomedical experts to transparently use a Grid infrastructure via high-level graphical tools. We have also provided bindings to access the EGEE infrastructure, within the same interfaces, thus enlarging the computational capabilities accessible. This way, a case study that analysis different ischemic conditions has been executed on the resources of the Biomed VO, using GMarteGS, obtaining a considerable speedup.

The widespread adoption of Grid Computing technologies require high level components that reduce the gap from the complexity of the underlying Grid middleware to the user. The development of a system, such as the one described in the paper, paves the way to interoperable metaschedulers which may cooperate via standard interfaces to provide resource brokering functionality on computational Grids.

References

[1] I. Foster, C. Kesselman, The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann, 2004.

[2] The Globus project: a status report, Future Generation Computer Systems, 15 (5-6), 607-621.

[3] I. Foster, Globus Toolkit version 4: Software for service-oriented systems., in: LNCS (Ed.), IFIP International Conference on Network and Parallel Computing, 3779, 2005, 2-13.

[4] LCG - LHC Computing Grid, URL: http://www.cern.ch/lcg

[5] J. M. Schopf, Ten Actions When Superscheduling, SchedWD 8.5, Scheduling Working Group (2001).

[6] J. M. Alonso, V. Hernández, G. Moltó, An object-oriented view of Grid computing technologies to abstract remote task execution, in: Proceedings of the Euromicro 2005 International Conference, 2005, 235-242.

[7] J. M. Alonso, V. Hernández, G. Moltó, GMarte: Grid middleware to abstract remote task execution, Concurrency and Computation: Practice and Experience 18 (15) (2006) 2021-2036.

[8] J. M. Alonso, J. M. Ferrero (Jr.), V. Hernández, G. Moltó, M. Monserrat, J. Saiz, Three-dimensional cardiac clectrical activity simulation on cluster and Grid platforms, Lecture Notes in Computer Science 3402 (2005) 219-232.

[9] J. M. Alonso, V. Hernández, R. López, G. Moltó, A service oriented system for on demand dynamic structural analysis over computational Grids, Lecture Notes in Computer Science 4395 (2007) 13-26.

[10] I. Foster, et al., The Open Grid Services Architecture, version 1.5 (March 2006).

[11] B. Sotomayor, L. Childer, Globus Toolkit 4: Programming Java Services, Morgan Kaufmann, 2005.

[12] WSRF - The WS-Resource Framework, URL: http://www.globus.org/wsrf.

[13] N. Ferguson, B. Schneier, Practical Cryptography, John Wiley & Sons, 2003.

[14] G. von Laszewski, I. Foster, J. Gawor, P. Lane, A Java commodity Grid kit, Concurrency and Computation: Practice & Experience 13 (8-9) (2001) 645-662.

[15] J. M. Alonso, J. M. Ferrero (Jr.), V. Hernández, G. Moltó, M. Monserrat, J. Saiz, Computer simulation of action potential propagation on cardiac tissues: An efficient and scalable parallel approach, Parallel Computing: Software Technology, Algorithms, Architectures and Applications, included in series: Advances in Parallel Computing 13 (2004) 339-346.

[16] B. Rodriguez, N. Trayanova, and D. Noble, Modeling cardiac ischemia, Ann. N.Y. Acad. Sci., vol. 1080 (2006) 395-414.

[17] EGEE - Enabling Grids for E-sciencE, URL: http://public.eu-egee.org.

[18] E. Huedo, R. S. Montero, I. M. Llorente, A modular meta-scheduling architecture for interfacing with pre-WS and WS Grid resource management services. Future Generation Computer Systems 23 (2) 252-261

[19] I. Rodero, J. Corbalan, R. Badia, J. Labarta, eNANOS Grid resource broker, Advances in Grid Computing - Egc 2005 3470 (2005) 111-121.

[20] E. Elmroth, J. Tordsson, An interoperable standards-based Grid resource broker and job submission service, in: e-Science 2005. First IEEE Conference on e-Science and Grid Computing, 2005, 212-220.

[21] W. Xiaohui, D. Zhaohui, Y. Shutao, H. Chang, L. Huizhen, CSF4: A WSRF compliant meta-scheduler, The 2006 World Congress in Computer Science, Computer Engineering, and Applied Computing (GCA'06), 2006, pp. 61-67.

[22] S. Venugopal, R. Buyya, L. Winton, A grid service broker for scheduling e-science applications on global data grids, Concurrency and Computation: Practice & Experience 18 (6) (2006) 685-699.