COMPONENT MEASURABLE VALUES AND SERVICES: A TECHNOLOGY FOR THE CONCLUSION OF RESOURCE TRANSACTIONS

Natalia Currle-Linde[1], Christian Pérez[2], Michael Resch[1], Massimo Coppola[3]

(1) High Performance Computing Center Stuttgart (HLRS),
University of Stuttgart, Nobelstrasse 19, 70569 Stuttgart, Germany
linde@hlrs.de, resch@hlrs.de

(2) INRIA/IRISA,
Campus de Beaulieu, 35042 Rennes cedex, France
Christian.Perez@inria.fr

(3) University of Pisa/CNR-ISTI,
Largo Bruno Pontecorvo, 3, 56127 Pisa, Italy
Massimo.Coppola@unipi.it

Abstract The absence of dynamic parameters in the description of software components does not allow the realization of traditional market relations for the Grid, based on transparency in the mechanisms of the allocation of Grid resources for customers and providers. This paper deals with the introduction into the technology of resource distribution of an apparatus of dynamic time and success estimation for the execution of concrete tasks. We discuss extensions to the GCM component model to include dynamically measured properties, the services needed to gather and convey the information, and their implementation using the distributed directory service from the XtreemOS project. With such a strategy it will be possible to realize a full set of traditional market relations and provide a flexible, efficient and transparent resource allocation.

Keywords: Grid, Software Component, Attribute, Services

1. Introduction

The Grid Component Model (GCM) [2] aims at mastering both the complexity of application and resources. However, so far there has been little effort to try to achieve an efficient and automatic deployment of GCM applications on Grids. The GCM Architecture Description Language (ADL) is not intended to be easily edited by end users, as it contains information about the mapping of components onto resources. Such a mapping requires a high level of expertise on applications and on resources.

To support an efficient execution of complex scientific or engineering applications, an adequate organization of resource distribution services is needed. Existing organizations of available services do not support reliable relations between individual clients and those who represent their interests, i.e. the Grid services, because these services are aimed at the simultaneous realization requirements of a multitude of clients and of the owners of the Grid resources. As there are no high-level instruments available for the automated planning of the execution of complex applications, it is generally not possible to provide services of an appropriate quality-level for these applications.

As service and/or resource failures are inherent properties of grids, there must be a control and a monitoring of the applications and of the resources, so that the application to be executed can at every moment achieve a high level of productivity. An example of such organization has been proposed in [1] in order to design a deployment model for Grids. However, that approach does not take into account dynamic properties that can be attached to a component.

This paper deals with dynamic component properties, and their use in the execution of complex applications in Grid organizations to the establishment of traditional market relations between users (clients) and owners of Grid resources. In particular, we propose the notion of *measurable components* as a foundation technology. An example of the needed supporting services for the notion can be found in the distributed directory service in development as part of the XtreemOS research project [5]. Section 2 introduces the notion of measurable components and their properties, while Section 3 describes the services managing the property values. Based on this, Section 4 focuses on the organization of the Grid with respect to a time property. Before the conclusion, Section 5 describes the interaction between applications and Grids.

2. Component Measurable Properties

There are several situations that need to deal with the description of a component. The main example of such a situation is the deployment phase, that selects a particular component implementation as well as the resource on which it will be deployed. Existing deployment tools only focus on a static description of a component, which takes into account properties like the architecture of the processor the implementation has been compiled to, the operating system, the implementation language and framework, etc. Dynamic properties like memory consumption, wall-time, etc. are seldom taken into account, but they are often essential to perform a relevant choice of deployment. For example, in a market relation, the user may impose economical constraints on the resources, like relating the cost of resources with their reliability and performance.

We believe it is important to propose a general solution, enabling static and dynamic properties to be attached to a component, and imposing no a priori restrictions on the set of properties used to direct deployment and adaptivity of components. Hence, we propose a model that extends standard component models with the notion of measurable property.

Component Measurable Property. The proposed model deals with component definition, extending the classical component definition with the notion of measurable property. A measurable property can be thought as a standard attribute but with the specificity that its value, which may be determined at runtime, needs to be dynamically retrieved and saved. Measurable properties can be attached both to primitive and to composite components.

Component Measurable Property Services. The model also comprises the definition of required services to manage measurable properties. There are three essential services, used to retrieve values (fetched from several component instances), to store them (preserve and organize information) and to access them (locate, apply policies, provide query functionalities).

The storage and access of information can in principle be solved by standard database technologies, but centralization issues would quickly hinder scalability over large Grids. Measuring values relies on the accurate definition of *when* these have to be read, and *who* should measure them. The framework should be responsible for measuring values like the memory consumption, while a component may be responsible of measuring specific properties closely related to its implementation. Services shall be generic with respect to *what* values have to be stored, if we do not want to constrain application performance models in advance. Under production settings, it is also important and not trivial to define *who* is allowed to access previously measured values.

This section focuses on the proposed model which deals with measurable properties for components. Before defining such properties and describing how GCM may be extended to support them, it first recalls some GCM features.

2.1 GCM **component description**

In most component models, components are statically described by the set of ports they expose, possibly by their implementation, and by the initial value

```
public interface anAttributeController extends AttributeController
{
    public String getValue();
    public void setValue(long value);
}

GCM ADL:
<definition name="AComponent" >
  <content class="AComponent"/>
  <attributes signature="AComponentAttributeController">
    <attribute name="Value" value="10"/>
  </attributes>
</definition>
```

Figure 1. Example of GCM attribute-controller definition and configuration.

of attributes. In GCM [2], a component definition may extend another component definition. It is possible to obtain different implementations of a component type by sub-typing its definition. A component can either be defined dynamically by means of the GCM API, or it can be defined statically thanks to the GCM ADL [2].

A component definition may also reference controllers, and it may define attributes that are dealt with by a specialized *attribute-controller*. The attribute-controller interface (see the example interface in Figure 1) must exhibit a setter/getter behavior with respect to an attribute. Attributes can also be configured in the ADL as shown in Figure 1 for the `Value` attribute.

Dynamic component description. From our viewpoint, static information is not enough to accurately describe a component. Dynamic information about a component is also needed for component selection (either at deployment time or at connection time), as for example its execution time or the amount of memory used, and, conversely, this information may depend on the resource selected for deployment. Hence we need a mechanism to describe values that need to be dynamically measured.

A component may export the values related to its dynamic behavior through attributes. This is a straightforward technique, but it suffers from two drawbacks. First, attributes are a general mechanism that targets component *configuration*: not all of them refer to a dynamic property of a component. Second, component implementations have to provide the implementation of the interface related to all attribute-controllers, while some measurable properties may only exist and be relevant while executing the component instance on a specific resource. It seems difficult to add an unplanned property to an *existing* component, thus attributes are an interesting but insufficient mechanism.

```
public interface MeasurablePropertyController {
    any getProperty(String property_name)
         throws IllegalPropertyException;
    List<String> getPropertyList();
}
```

Figure 2. Interface of a controller to retrieve any component property.

2.2 Measurable property definition

We define a *measurable property* as a value associated to a component instance that contributes to characterize it. The values of a measurable property obtained from two instances of the same component may be different as the value is *measured* on a component instance. Measurable properties are usually expected to be dynamically retrieved, and to depend on the computational resource where the component instance is executed.

Measurable properties that are associated to a component can be provided either by the component itself or by a framework. In the former case, properties are said to be *internal*, while in the latter they are said to be *external*. External properties are values which are more easily or conveniently computed outside the component, such as the memory or the bandwidth consumption.

Therefore, a mechanism is needed to associate a measurable property with its producer. We envision two different kinds of property producer. The simplest case is when an attribute provides the property. The second case occurs when the property is retrieved through an interface. These two situations are detailed below.

Internal properties are directly provided by the component itself. Hence, it appears straightforward to re-use the attribute mechanism to get them. As such values are read-only, only a getter method is required in the attribute-controller.

External properties are provided by the framework. Thus, the framework has to provide the values for *several* components. As the framework might not be a component, we may only expect that it will provide an interface. With respect to the GCM design philosophy, such an interface shall be exported through a component controller. Figure 2 gives an example of the API of such a controller. It is important to note that the implementation of such a controller must be under the responsibility of the framework, not of the component.

When taking into account both internal and external properties, it is obvious that they can be combined into a coherent interface. With respect to the outside of the component, a unique mechanism is needed. Hence, a compliant component implementation shall provide a controller whose interface is of type `MeasurablePropertyController`. Its implementation shall be able to directly return the values of internal properties and it shall invoke the framework provided controller for external properties.

As shown in Figure 2, the `MeasurablePropertyController` interface also has an operation that returns the list of all properties available so as to cope with the introspection property of GCM. In general, it is not possible to statically know the list of properties that will be available once a component is going to be deployed. We decide to support the more general case. If needed, component definition can be extended to enforce the list of supported properties, but internal and external properties have to be differentiated so as to check whether the framework a component is going to be deployed to supports the requested external properties.

3. Component Measurable Property Services

According to the viewpoint of the CoreGRID Institute on Grid Systems, Tools and Environments, a generic grid platform is made up by a set of components, which typically are GCM components. Together, they provide specialized services like resource discovery, file transfer, and job launching. The set of services needs to be extended to manage measurable features of component outlined in Section 2.

In the following we describe how these services are structured, a possible implementation based on the Directory Service for Services and Resources developed within the XtreemOS project [5], and we analyze how it would be possible to integrate these services within the GCM framework.

Value Retrieving Service. The Value Retrieving Service (VR) is responsible to retrieve the property values associated to component instances. Hence, it should know the list of component instances, *which* values has to be retrieved and *when* they have to be retrieved.

In GCM, a component is created thanks to the `newFcInstance` operation of the `GenericFactory` interface. Such an operation typically relies on an application deployment component. Hence, either a `GenericFactory` or a deployment component has to inform the VR service that a new component has been created. The first problem is thus solved by defining a dependency between the VR and the deployment service.

A component dynamically declares its measurable values (as defined in Section 2), thus solving the second problem, as the VR service can obtain the list of measurable properties from all instanced components.

Even simply assuming that a property value (possibly a list of values) is retrieved at the end of the component execution, the GCM specification does not include an operation to destroy a component, i.e. the inverse of the `newFcInstance`.

In order to provide a portable retrieval mechanism for measured properties, we advocate an extension of the GCM API with a `destroy` operation. Such operation could be added to the `GenericFactory`. Under these assumptions,

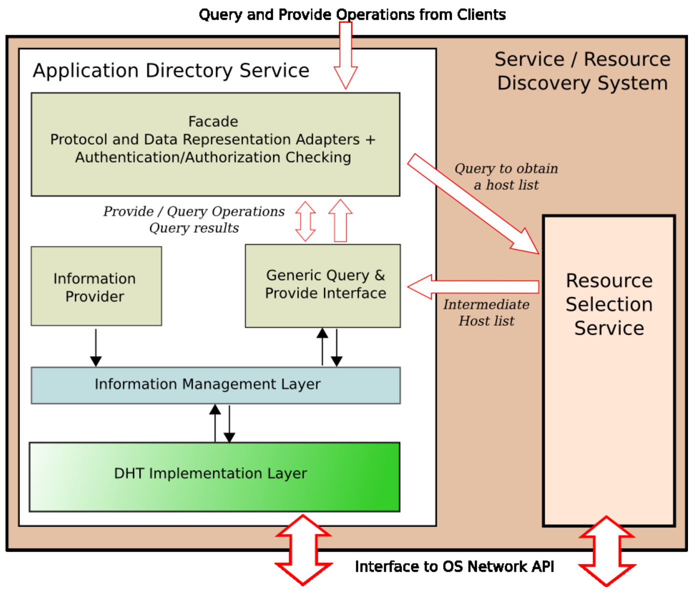
Query and Provide Operations from Clients
Application Directory Service
Service / Resource Discovery System
Facade
Protocol and Data Representation Adapters +
Authentication/Authorization Checking
Query to obtain a host list
Provide / Query Operations
Query results
Information Provider
Generic Query & Provide Interface
Intermediate Host list
Resource Selection Service
Information Management Layer
DHT Implementation Layer
Interface to OS Network API

Figure 3. High-level software architecture of the Service and Resource Discovery System.

the VR service can be implemented by component interfaces in the measured components and in the deployment structure.

Storing Service. To be able to usefully exploit the measured properties we explicitly assume that all component types and component implementations are uniquely identified, so that the collected data is never ambiguous. Besides, the Storing Service SS will also need to identify all the computing resources uniquely (which is a simpler problem solved in term of public and private IP addresses) in order to distinguish e.g. performance obtained on different systems by the same component implementation.

The SS manages collected property values, and inherits all the requirements of the VR. Its design must avoid centralization points, as the SS has to be inherently distributed on the Grid platform, to control a large amount of component instances belonging to different applications, and it has to concurrently collect and deliver information for multiple users.

Very similar requirements and properties were devised within the XtreemOS project[1] for the Service/Resource Discovery System (SRDS, [5]).

The SRDS, whose node-level architecture is shown in Figure 3, is designed to be a pervasive support, to store and retrieve a large amount of information,

[1]Project no. IST-033576 XtreemOS - "Building And Promoting A Linux-Based Operating System To Support Virtual Organizations For Next Generation Grids".

satisfying requirements of efficiency, high availability and reliability over Grid platforms composed of tens of thousands of machines. An instance of the system is deployed on each computation node in XtreemOS, its network interfaces participating in the SRDS overlay networks.

The SRDS is a flexible directory service, supporting different data formats and query functions. It provides basic storage and retrieval functionality for **key-value** pairs with generic values (e.g. small files, attribute lists), as well as more refined query forms (e.g. **range queries**, neighborhood queries), that are supported when a specialized semantics has been defined on the data values. Simple and complex queries can be performed on key-value pairs whose values are **dynamically updated**, e.g. handling real-time measured quantities and properties for the sake of monitoring or to support decision making.

The SRDS exploits a combination of peer-to-peer, unstructured and Distributed Hash-Tree overlay networks [6]. Each hosting node belongs to one or more of them. The multi-stage query execution approach provides a flexible set of churn-tolerant services, exploited by several modules of the XtreemOS system, including the Deployment System, and by the applications. Thus the SRDS can provide scalable support also in the framework of a component system.

Access Service. The Access Service (AS) has to provide efficient access to the measured values, but also mechanisms for access control and restrictions, that are needed for a market-aware use. For instance, it will often be the case that measured properties concerning a component implementation, a set of resources or users, have to be kept private to a specific Virtual Organization (VO), or role within the organization.

Within XtreemOS, the Facade module in the SRDS architecture (see Fig. 3) interfaces to XtreemOS node-local VO services in order to authenticate and authorize all provides and queries with respect to the VO service policies. In order to gather measurable values from GCM components, the Facade has instead to include mechanisms allowing the GCM framework and its components to dynamically register with the SRDS. The mechanisms needed are described in the following Sections 4 and 5, and can be implemented as Facade plug-ins, with no change to the rest of the SRDS architecture.

4. Time features of application programs and the model for the organization of the Grid.

So far industrial Grid systems have not yet solved two important problems: devising a general and efficient methodology for the distribution of tasks for each concrete application, and efficiently planning the execution of complex Grid applications.

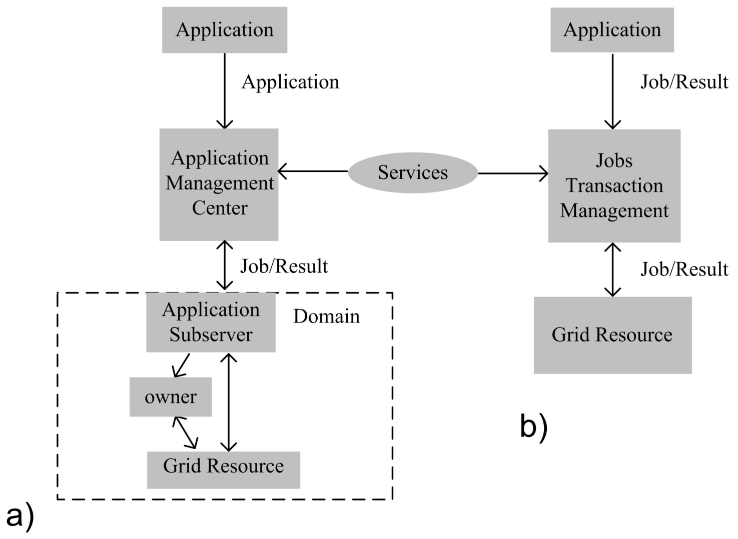

Figure 4. Model for the organization of the Grid

We want to address a major issue with respect to these problems, the general difficulty to know in advance the execution time for each application task. We will call this the problem of "inexact time". We do not address however, the research on parallel performance models and techniques to derive execution parameters from bottom-up synthesis of analytic models.

We will instead exploit time-related measurable properties such as the known times required for the execution of programs on a certain set of machines, and eventually use them to approximate top-down the time depending on other parameters, like the number of available processors, the volume and quality (e.g. accuracy) of the data to be processed. The introduction of these resource-dependent and application-specific parameters into the set of properties characterizing application components, makes it possible to improve the efficiency of Grid applications.

In a methodology addressing application behavior estimation, it is necessary to determine the time attributes during the development of application programs for the Grid. A prerequisite is that all application programs are thoroughly verified and tested in order to guarantee their quality, and avoid e.g. resource configuration related bugs. The lack of such a quality control procedure is one reason slowing down Grid adoption in business and industry.

To allow enhancing the planning of application executions, besides introducing dynamic measurable component properties, a modernization of Grid infrastructure is needed, to incorporate the related management services. Figure 4 shows two types of Grid organization. The first one is the one we propose to achieve a better performance for complex applications. It prioritizes the maximum efficiency of execution for a single complex application. (see Figure 4(a)). The second one aims at maximum utilization and throughput performance of Grid resources. This is currently standard model (see Figure 4(b)).

As soon as management centers with the newly proposed approach have received the complete information about the resources required for one complex application at each stage of computation (including economic parameters), the execution can be planned with high efficiency. At the same time it will be possible to make corrections in the plan during the execution of an application depending on the current state of each Grid resource.

5. Interaction between applications and the Grid.

We address the efficient execution of complex Grid applications not only by solving the issue of elementary Grid jobs, but also to with an appropriate Grid organization and with tools for the automated planning of the application execution exploiting a universal economical scheme of the distribution of resources. We will introduce a method for the organization of the management of complex applications which is based on the problem solving environment SEGL [3].

The planning process covers the whole experiment, from the design and development of an application, through its execution and up to its final completion. At the beginning of the design phase, i.e. in the verification phase, there will be an evaluation of the minimum number of resources required for the execution, taking into account user-specified times for the application components. For this purpose it is necessary to model the execution of the application. Two goals have to be pursued. First it will be checked whether the execution of the program is correct. Second it will be necessary to determine the maximum execution time allowed for each block (as used in SEGL [3]) of the application and the minimum resource requirements for their execution. In addition the total number of resources required at each time will be calculated, taking into account the level of parallelization the application permits. During the design phase the possible cost limits for the resources required for each block will also be calculated.

In the following phase, planning is taken over by the Management Center. The Deployment Model of the Application Manager makes use of various types of information services, and creates a pool of domains and of individual candidate machines which are best suited for the execution of applications. The selection of candidate machines is carried out on the basis of an analysis of the potential capacities of the Grid domains as well as an analysis of the current states of the resources, of their availability, their reliability (probability of failure and deadline missing) and of their costs. A list of candidate machines most suited for the execution of each program block of the application,

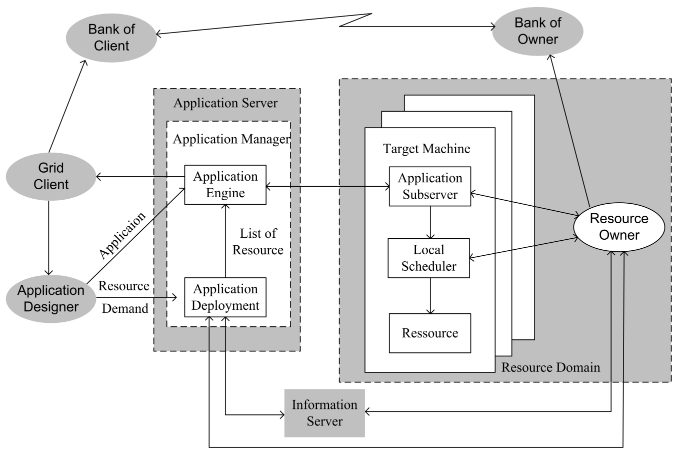

Figure 5. Grid organization within an economical scheme of distribution of resources

as well as a list of the Grid resources for the execution of the application, will be generated.

The management center, exploiting the annotated pool of resource, concludes contracts on behalf of the users with the owners of the resources. During this phase the necessary legal and bank accounting information is exchanged between, and examined by the two parties, before the final list of candidate machine and domains is generated. After this the sub-server programs which represent the interests of the managing center of the application are sent to the selected machines and domains. These sub-server programs are responsible for the local monitoring of the current state of resources. They act as agents during the conclusion of current and future resource transactions between the application and the owners of the resources. They also manage the execution of application jobs.

During the execution phase different scenarios and policies are possible in the interaction with resources. In case a program block is executed in SEGL batch mode the reservation of resources and the conclusion of a contract is only carried out once for the whole block. This is done taking into account the performance required for the execution of all jobs of the block within a certain time. The resources can be reserved on one or on several machines belonging to different administrative domains.

In the case of the SEGL pipeline mode for a chain of blocks, provided the execution of a job requires much time, an appropriate variant would be the successive reservation of resources and the successive conclusion of contracts each time the next job has been prepared for execution. This variant has been described in more detail in [4].

As in the course of the execution, the working condition of resources required is constantly monitored, the decision concerning their reorganization in the event of a failure is taken on the spot. In such a case the old resource transaction is canceled and a new set of resources is generated to start a new job. Payment of resource transactions is affected accordingly, and the conclusion and payment of transactions are controlled by the banks of both parties, the resource owners and the clients. The organization of the interaction between an application and the Grid in the development phase is shown in Figure 5.

6. Conclusion

We have proposed an extension of the GCM with measurable properties, to be managed by a combination of framework-provided interfaces and distributed retrieval services. The provided information about dynamic properties of component instances over different resources is leveraged in a testing and execution environment aiming at solving the "inexact time" problem. The approach allows the introduction into the Grid of traditional and transparent economic mechanisms for the resource allocation, both for the clients and for the resource owners.

The introduction into the Grid of an apparatus of management centers enables the realization at a high organizational level of an optimal planning and efficient control for complex Grid applications generated in different areas of science, industry and business.

References

[1] Massimo Coppola, Marco Danelutto, Sébastien Lacour, Christian Pérez, Thierry Priol, Nicola Tonellotto, and Corrado Zoccolo. Towards a common deployment model for grid systems. In Sergei Gorlatch and Marco Danelutto, editors, *CoreGRID Workshop on Integrated research in Grid Computing*, pages 31–40, Pisa, Italy, November 2005. CoreGRID, IST.

[2] Programming Model Institute. Basic features of the grid component model (assessed). Technical report, CoreGRID, March 2007. D.PM.04.

[3] Currle-Linde, N., Küster, U., Resch, M., Risio, B.: *Science Experimental Grid Laboratory (SEGL) Dynamical Parameter Study in Distributed Systems*. ParCo 2005, Malaga, Spain, 2005.

[4] N. Currle-Linde, P. Adamidis, M. Resch, F. Bös, J. Pleiss: *GriCoL: A Language for Scientific Grids*, Proceedings of the 2nd IEEE International Conference on e-Science and Grid Computing, pp. 62, Amsterdam, December 2006.

[5] M. Baldanzi, M. Coppola, P. Costa, D. Laforenza, G. Pierre, and L. Ricci *Design and Specification of a Prototype Service/Resource Discovery System* XtreemOS Technical Deliverable D3.2.4, Work Package 3.2, November 2007.

[6] Antony Rowstron and Peter Druschel. Pastry: Scalable, Decentralized Object Location and Routing for Large Scale Peer-to-Peer Systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, 2001.