Deploying CORBA Components on a Computational Grid: General Principles and Early Experiments Using the Globus Toolkit

Sébastien Lacour, Christian Pérez, Thierry Priol

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Deploying CORBA Components on a Computational Grid: General Principles and Early Experiments Using the Globus Toolkit

Sébastien Lacour, Christian Pérez, Thierry Priol

N°5148

March 25th, 2004

THÈME 1

ISSN 0249-6399 ISRN INRIA/RR--5148--FR+ENG

Deploying CORBA Components on a Computational Grid: General Principles and Early Experiments Using the Globus Toolkit

Sébastien Lacour, Christian Pérez, Thierry Priol*

Thème 1 — Réseaux et systèmes
Projet Paris

Rapport de recherche n°5148 — March 25th, 2004 — 19 pages

Abstract: The deployment of high bandwidth wide-area networks has led computational grids to offer a very powerful computing resource. In particular, this inherently distributed resource is well-suited for multiphysics applications. To face the complexity of such applications, the software component technology appears to be a very adequate programming model. However, to take advantage of the computational power of grids, component-based applications should be *automatically deployed* in computational grids. Based on the CORBA component specifications for the deployment of components, which seem to currently be the most complete, this paper proposes a detailed process for component deployment in computational grids. It also reports on early experiments on deploying CORBA components in a computational grid using the Globus Toolkit 2.4.

Key-words: CORBA components, CCM, component deployment, computational grid, Globus Toolkit.

(Résumé : tsvp)

Citation: this paper has been published in the *Proceedings of the 2nd International Working Conference on Component Deployment* (CD 2004), held in conjunction with the 26th International Conference on Software Engineering (ICSE 2004), Edinburgh, Scotland, UK, May 2004, Springer-Verlag, LNCS. Please, mention this reference for any citation.

* {Sebastien.Lacour,Christian.Perez,Thierry.Priol}@irisa.fr

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)
Téléphone : 02 99 84 71 00 - International : +33 2 99 84 71 00
Télécopie : 02 99 84 71 71 - International : +33 2 99 84 71 71

Déploiement de composants CORBA sur une grille de calcul: principes généraux et expériences préliminaires avec Globus

Résumé : La mise en place actuelle de réseaux longue distance à forte bande passante permet aux grilles de calcul de se présenter comme une ressource de calcul très puissante. En particulier, cette infrastructure, distribuée par nature, est bien adaptée pour l'exécution d'applications multiphysiques. Pour faire face à la complexité de telles applications, les composants logiciels apparaissent comme un modèle de programmation tout à fait adéquat. Cependant, pour pleinement profiter de la puissance de calcul des grilles, les applications à base de composants doivent pouvoir être *déployées automatiquement* sur de telles infrastructures. À l'heure actuelle, les spécifications pour le déploiement de composants CORBA semblent être les plus abouties. En se fondant sur le modèle de déploiement de composants CORBA, cet article propose un processus détaillé de déploiement de composants sur une grille de calcul. Ce document fait aussi état de nos expériences préliminaires en matière de déploiement de composants CORBA à l'aide de Globus 2.4.

Mots-clé : Composants CORBA, CCM, déploiement de composants, grille de calcul, Globus Toolkit.

1 Introduction

Networks have been growing very quickly for several years in terms of bandwidth, as stated by Gilder's law claiming that network bandwidth increases much faster than processors' power. The bandwidth of the currently deployed wide-area networks has reached a level allowing several distributed computing resources to be used together within a grid to handle complex applications in various domains such as scientific computing. In particular, grid infrastructures are very valuable for the execution of new scientific applications which require a huge amount of computing power or physical resources which do not exist in a single physical location. Multiphysics applications, combining several simulation codes, are examples of such new demanding applications. They aim to simulate complex phenomena which require the simulation of various physics: computational fluid dynamics and structural analysis can be combined to improve the design of aircrafts; simulating the behavior of the ocean jointly with the atmosphere can provide more accurate weather forecast. They are just a few examples of a growing number of coupled applications. Usually such applications are programmed with tools designed for parallel computing (like MPI). Several *ad-hoc* code couplers have been developed to couple simulation codes together and to run them on either parallel machines or grid infrastructures. However such an approach lacks genericity and requires each simulation code to be adapted to a specific code coupler. Modularity and encapsulation are provided at their lowest level, making the re-usability of codes very limited. Programming models allowing assembly and re-usability of pieces of software have been investigated for some time. However, few attempts have been made to apply such models to scientific computing either due to the lack of efficiency of these models or the quasi absence of appropriate software environments (including deployment) on high-performance computers. It is even more acute in grid infrastructures.

State of the art work in programming models suggests that programming models based on the software component model [29] provide many benefits. For instance, the component model emphasizes the development of an application by assembling (composing) existing pieces of software. Hence, not only is the development time reduced, but code can be reused and the resulting application presents more modularity. Several component models are available like Enterprise Java Beans (EJB) by SUN [27], Web Services [4], the CORBA Component Model (CCM) [22] or the Common Component Architecture (CCA) [1]. In particular, these models differ with respect to deployment. Web Services as well as CCA do not specify how components should be deployed, while EJB and CCM do. The CCM deployment model supersedes the model proposed by EJB because CCM allows for the deployment of a set of interconnected components on a set of distributed machines. Thus, we choose to work with CCM as it presents the most complete deployment model. In previous work, we have shown that high performance communication is possible using CORBA [8] and that high performance computing can be brought to CORBA by extending its specification with the concept of parallel component [25]. The results obtained from this research show clearly there is no obstacle in using CORBA for high-performance computing.

As already mentioned, computational grids can yield a potentially huge computational power, federating resources spread all over the world [10]. However, access to this computational power has a price. A computational grid can consist of very heterogeneous resources, such as compute nodes with different architectures and operating systems, networks with various performance characteristics, storage resources of different sizes, *etc.* Reliable measures must be taken to enforce security, involving authentication, authorization, data encryption for confidentiality, *etc.* Thus, a middleware is necessary to solve those issues so that users can actually, easily access computational grids. The Globus Toolkit [34, 9] is such a wide-spread and well-established middleware, as exemplified by TeraGrid [40].

The remaining issue is to deploy components in a grid environment. Our objective is to be able to *automatically* deploy CORBA components in a computational grid, thus marrying the advantages of both worlds: distributed computing provides the ease of programming using CORBA components while grid computing provides the tools to effectively, securely deploy applications in computational grids. The *actual* deployment of components is essential for the success of the component programming model, and it must be as *automatic* as possible for the wide acceptance of the model. Automatic deployment means that the client should not have to worry about the compute nodes which will execute the component-based application. Ideally, the client should not have to be a computer scientist, but a physicist, a chemist, a biologist, *etc.* Thus the deployment phase should be very simple and merely require a pointer to a computational grid which the client has access to plus a packaged component-based application.

This paper presents how we envision the deployment process of CORBA components on a computational grid managed by the Globus Toolkit and mentions a number of issues encountered at deployment time. Section 2 introduces the CORBA component model. The Globus Toolkit is presented in Section 3. Section 4 deals with the deployment of CORBA components in a grid: it shows how both software components from the world of distributed computing and grid access middleware from the world of grid computing can be used in combination. The early experiments we carried out are detailed in Section 5. Before concluding, some related works are presented in Section 6.

2 Overview of the CORBA Component Model

The CORBA Component Model [22] (CCM) is part of the latest CORBA [23] specifications (version 3). The CCM specifications allow the deployment of components into a distributed environment, that is to say that an application can deploy interconnected components on different heterogeneous servers in one operation. Figure 1 presents the general picture of CCM. The component life-cycle is divided into two parts. First, the creation of the component requires to define the component interface, to implement it and then to package it so as to obtain a component package, *i.e.* a component. The second part consists in (optionally) linking together several components into a component assembly and in deploying it. CCM provides a model for all theses phases. For example, the CCM abstract model deals

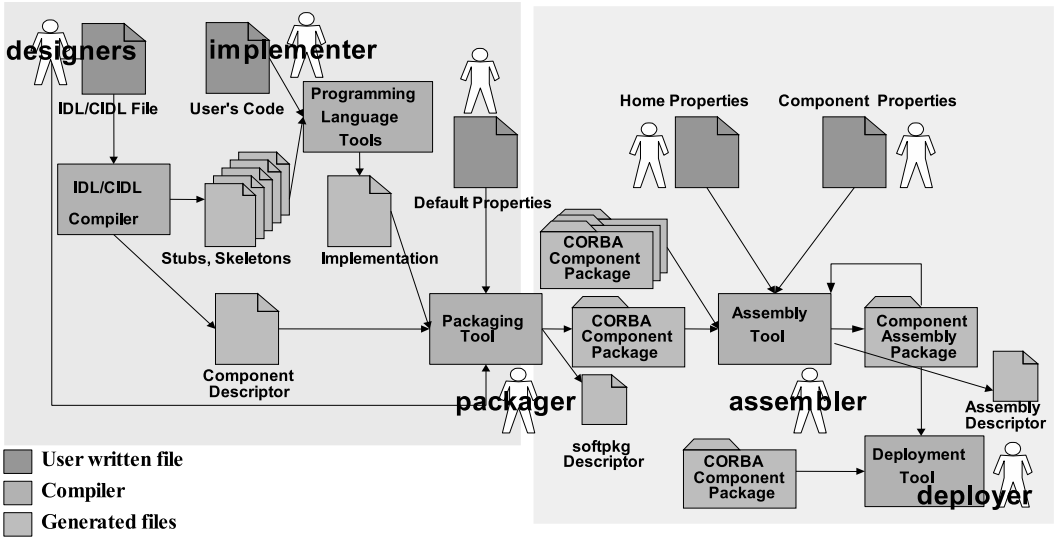

Figure 1: Overview of the CORBA Component Model.

with the external view of a component, while the Component Implementation Framework (CIF) provides a model to implement a component. There are also models for packaging and deploying a component, as well as for the runtime environment of a component. In this section, we briefly introduce the abstract model, the execution model and the deployment model.

2.1 CCM Abstract Model

A CORBA component is represented by a set of ports described in the Interface Definition Language (IDL) of CORBA 3 defined by the OMG. The IDL of CORBA 3 is an extension of the IDL of CORBA version 2 by the OMG. There are five kinds of ports as shown in Figure 2. *Facets* are named connection points that provide services available as interfaces while *receptacles* are named connection points to be connected to a facet. They describe the component's ability to use a reference supplied by some external agent. *Event sources* are named connection points that emit typed events to one or more interested event consumers, or to an event channel. *Event sinks* are named connection points into which events of a specified type may be pushed. *Attributes* are named values exposed through accessor (read) and mutator (write) operations. Attributes are primarily intended to be used for component configuration, although they may be used in a variety of other ways. Figure 3 shows an example of component definition using IDL3.

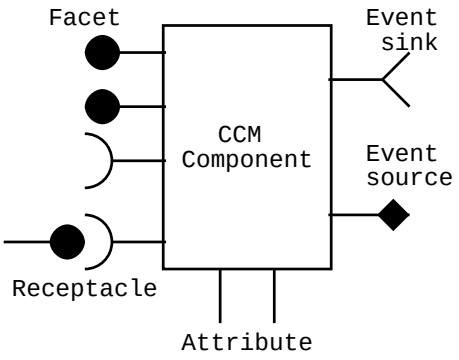

Figure 2: A CCM component.

```
// Interface Average definition
typedef sequence<double> Vector;
interface Average {
  double compute(in Vector v);
};
// Component A definition
component aComponent {
  attribute string  name;
  provides  Average avgPort;
  uses      Display dspPort;
};
```

Figure 3: A component IDL definition.

```
aComponent ref = ServerComp->provide_avgPort();
ClientComp->connect_avgClientPort(ref);
```

Figure 4: Example of code to connect two components.

Facets and receptacles allow a synchronous communication model based on the remote method invocation paradigm. An asynchronous communication model based on data transfer is implemented by the event sources and sinks.

A component is managed by an entity named *home*. A home provides factory and finder operations to create and/or find a component instance. For example, a home exposes a `create` operation which locally creates a component instance.

2.2 CCM Execution Model

CCM uses a programming model based on containers. Containers provide the run-time environment for CORBA components. A container is a framework for integrating transactions, security, events, and persistence into a component's behavior at runtime. Containers provide a standard set of services to a component, enabling the same component to be hosted by different container implementations. All component instances are created and managed at runtime by its container.

2.3 CCM Deployment Model

The deployment model of CCM is fully dynamic: a component can be dynamically connected to and disconnected from another component. For example, Figure 4 illustrates how a component **ServerComp** can be connected to a component **ClientComp** through the facet **FacetExample**: a reference is obtained from the facet and then it is given to a receptacle. Moreover, the model supports the deployment of a static application. In this case, the assembly phase has produced a description of the initial state of the application. Thus, a deployment tool can deploy the components of the application according to the description. It is worthwhile to remark that it is just the initial state of the application: the application can change it by modifying its connections and/or by adding/removing components.

The deployment model relies on the functionality of some fabrics to create component servers which are hosting environments of component instances. The issues of determining the machines where to create the component servers and how to actually create them are *out of the scope* of the CCM specifications.

3 The Globus Toolkit: a Grid Access Middleware

The Globus Toolkit [34, 9] is an open source software toolkit used for building grids. ("Globus Toolkit" is a trademark of the University of Chicago.) It includes software for security enforcement, resource, information, and data management, portability. This middleware is wide-spread and well-established, as exemplified by many projects relying on the Globus Toolkit, such as GriPhyN (the Grid Physics Network, [37]), the American DOE Science Grid [33], the European DataGrid project [32], TeraGrid [40]: see [31] for a list of software installed on TeraGrid machines, the *Common TeraGrid Software Stack* (CTSS), including the Globus Toolkit.

The Globus Toolkit uses the Grid Security Infrastructure (GSI, [11]) for enabling secure *authentication* and *communication* over the grid network. GSI is based on public key encryption (X.509 certificates) and the Secure Sockets Layer (SSL) communication protocol. Globus users can initialize proxies, which are certificates signed by the user: proxies provide a convenient alternative to constantly entering passwords when submitting many jobs.

The resource management module of the Globus Toolkit relies on GRAM (Globus Resource Allocation Manager) and GASS (Global Access to Secondary Storage). GRAM [7] supports remote job submission and control using GSI for mutual authentication of both users and remote resources: GRAM's authorization mechanism is based on GSI identities and a mechanism to map GSI identities to local user accounts (see the `grid-mapfile` configuration file of the Globus Toolkit). GRAM understands the RSL (Resource Specification Language, [26]) which provides an interchange language to describe grid resource requirements and to submit a job. GASS [2] provides basic access to remote files, allowing file staging before and after the execution of a job on a grid node.

The Grid Information Service (GIS) of the Globus Toolkit is implemented in MDS (Monitoring and Discovery Service, [6]). It includes robust authentication mechanisms using GSI. MDS allows for flexible, hierarchical, distributed (scalable), extensible information storage; it is based on an LDAP directory. MDS also supports dynamic sources of information, such as the Network Weather Service (NWS, [41, 28]).

Finally, the data management module of the Globus Toolkit provides the ability to access and manage data in a grid environment. It includes GridFTP [35, 36], which is used to move files between grid-enabled storage systems. GridFTP is a high-performance, secure (using GSI), reliable data transfer protocol optimized for high-bandwidth wide-area networks: it allows parallel transfers, third-party transfers, partial file transfers.

A grid middleware system like the Globus Toolkit provides features to access and control grid resources. It does *not* provide any programming model.

4 Deployment Process of CORBA Components in a Computational Grid

This section describes how we envision the general deployment process of CORBA components *in a computational grid*. This process complies with the deployment and configuration process [21] as specified by the Object Management Group (OMG, [39]).

4.1 Initial Assumptions

Before the deployment takes place, we assume that the computational grid has already been built and deployed on the computing resources. This is a reasonable hypothesis as we rely on the Globus Toolkit for its widespread adoption. The grid access middleware as well as the ORB have been installed on the resources of the grid.

We also assume that the user's authentication certificate has already been obtained, installed and signed by a certificate authority (CA) attached to the target computational grid. The user's grid identity must be mapped to a user identity (Unix username) on each resource of the grid. An authentication proxy must also have been initialized to access the computational grid resources.

The accurate resource description of the computational grid must already be compiled and available. This resource description can typically be included in the information service of the grid access middleware, such as the MDS of the Globus Toolkit. The resource description should be precise enough to describe compute node characteristics (architecture, memory size, operating system, CPU speed, *etc.*), network topology (interconnection links, firewalls, *etc.*) and performance properties (bandwidth, latency, jitter, loss), storage capacities, *etc.*

Finally, we assume that the component or component assembly to be deployed has already been packaged in a ZIP file: according to the CCM specification, component packages should have a `.zip` extension, and component assembly packages should have a `.aar` extension.

4.2 Starting Point

Ideally, *automatic* deployment should allow the client user to run a component-based application on a grid which he has access to by simply typing a command line in his terminal. This command (the "deployment tool", executed on a computer which we call the "deployment client") merely requires:

- a CCM component assembly package (`.aar`), or a component package (`.zip`), or a pointer (URL) to such a file;
- a pointer to the information service describing the resources available in the computational grid; in the Globus Toolkit's terminology, that would be a "base distinguished name" in the MDS tree.

This command should return a handle to steer (get status, suspend, restart, terminate) the component-based program's execution. The actions of this magic command are described in the following subsections.

4.3 Planning the CORBA Component Deployment

The goal of this phase is to determine on which compute nodes the component-based application will run. More precisely, it will determine which component will execute on which particular compute resource of the computational grid.

First, we need to extract information out of the archive (component or assembly package) provided on the command line or downloaded from the location given on the command line. According to the CCM specification, the archive is self-described, since it contains a package descriptor file (or a pointer to such a package descriptor). The component or assembly package contains deployment constraints such as the following.

Architecture and operating system. The component(s) in a package have been compiled for a specific computer architecture and under a specific operating system, so they will have to run on their target architectures under the proper operating system. A package may contain various components compiled for different architectures and operating systems.

Dependencies. A component may have environmental or other dependencies: the component deployment tool must make sure those dependencies will be satisfied at execution

time. The dependencies may be on libraries, executables, JAVA classes: if necessary, they must be staged in on the compute node running the component prior to execution.

Host or process collocation. The CCM specification allows the components of a component assembly to be partitioned: components may be free or partitioned to a single process or a single host, meaning that a group of component instances will have to be deployed in the same process or on the same compute node.

There is a possibility to enrich the constraints attached to a component package or a component assembly package using the `extension` XML element, yielding still more deployment tool-specific constraints. Those constraints could concern network technology requirements for performance, desired completion time, *etc.* but this is out of the scope of this paper.

A component assembly package may also contain various implementations (possibly using different programming languages) of the same component interface.

After gathering all the information pertaining to deployment plus the description of the grid resources, the second step consists in *automatically mapping* the component(s) to the computing resources where they will execute. This part is not the focus of this paper, and research has already been conducted on this topic [14, 13, 19] (*cf.* related works in Section 6).

4.4 Preparing Execution and Launching Processes on a Computational Grid

The previous phase resulted in a *deployment plan* which specifies which components must be launched on which nodes of the computational grid.

Using the grid access middleware, we stage in to the execution nodes the input data files and component dependencies (libraries, JAVA classes, *etc.*). Ideally, the automatic determination of the deployment plan should have selected nodes close to each other for communication efficiency. The locality of the selected nodes may allow various file staging strategies. For instance, instead of transferring files twice from a remote repository to two compute nodes close to each other, we may prefer to transfer files once from the remote repository to one of the compute nodes, which will then forward the files to the other node in close neighborhood. Depending on the information services describing the grid resources, we may be able to detect that two selected compute nodes share a common file system (*e.g.*, NFS): in this case, another possible strategy would consist in staging in input data to the common file system just once. Then both compute nodes could access the read-only input data through the shared file system.

Now the component servers can be launched as regular processes on the compute nodes selected in the deployment plan. Each component server returns a CORBA reference (IOR) to the deployment tool invoked from the deployment client: this IOR will be necessary to control the component server. Then CORBA containers get created within the component servers to install the CORBA homes objects, and the homes create the components as stated

by the deployment plan: all those newly created objects return IOR references for the deployment tool to be able to control the newly created entities.

If we want an actual control over the application, the IORs must be returned to the deployment tool using a secure channel and as soon as the objects which they reference get created. In particular, they must be returned before the application completion. This conversational mode of communication between the deployment client (which submits the job to execute) and the various processes of the running application is rather unusual in grid computing. Most often, grid computing applications do not need to exchange messages with the deployment client during execution: the input data is transfered all at once *prior to* execution, and the output data is retrieved *after* execution is completed. Of course, this does not mean that the subtasks of the application do not communicate with each other during execution.

This conversational mode is also necessary for the following reason. As we consider a grid environment, we do not own the resources we use, so we assume that each compute session (*i.e.*, each application deployment) is independent of all other sessions, should they be initiated by the same user or by another user of the computational grid. In practice, the component servers, containers, components, *etc.* register to a name service (this is out of the CORBA specification). This name service must be operational *before* the components get launched because they need the IOR of (or any sort of reference to) the name service. However, we cannot count on a persistent name service running for multiple compute sessions, and the name service should be deployed first by the deployment tool. Here again, we see that we have a conversation-like operation mode, where the deployment client launches a process on a compute node, this process answers with an IOR, and this answer triggers another action on the deployment client's side.

The location of the name service would be completely up to the deployment tool, but a reasonable strategy would choose a compute node close to as many compute nodes hosting the application components as possible.

4.5 Configuration and Start of Execution

The last phase of deployment consists in connecting together the components of an assembly and configuring the components. To do that, the deployment tool may need once again CORBA references (IORs) to the components which have been launched (or a reference to the name service if those components registered to it). Usually, client requests start the server components' execution.

4.6 Execution Control

Control means that the deployment tool can get the status (active, completed, *etc.*) of the running application, cancel, suspend or restart its execution. Thus the deployment tool can cleanly deallocate the resources it uses to run the application. This control may also be

useful for application re-deployment, or dynamic visualization component connection: those two issues are out of the scope of this paper. See [20] for algorithms which dynamically remap the subtasks of a running application.

The component servers are launched as regular grid processes. The latter return a handle to the grid access middleware, which is not aware that it is launching CORBA processes: this handle allows the grid access middleware to monitor and control its processes. Thus, the deployment tool has two ways to steer the application execution: the CORBA IORs plus the grid middleware handles for each executing process.

5 Early Experiments with MicoCCM and the Globus Toolkit

The grid access middleware we use is the Globus Toolkit version 2.4.3. We rely on this version since it is now well established and stable. We did not rely on version 3 of the Globus Toolkit because it lacks maturity and its future is currently uncertain. Version 4 of the Globus Toolkit is not available yet, but migration to this future version should not be an issue. The ORB we utilize is MicoCCM version 2.3.11, a fully compliant implementation of CORBA, including the CORBA component model (CCM).

As mentioned in Subsection 4.1, we assume that we have a valid, already packaged component or component assembly (using Frank Pilhofer's Assembly Toolkit [24]), as well as the list of hosts which will execute the components. Currently, we have no deployment planner implemented, so the resources are selected in a round-robin manner, as provided by the MDS. However, we acknowledge that deployment planning is a very important phase in a grid environment, and it should be designed carefully, since it can have a huge impact on execution performance. This work is unavoidable in a near future.

In our current implementation, input data files and component dependencies are staged in to each compute node directly from their repositories: the subtle strategies evoked in Subsection 4.4 have not been implemented yet, all the more so as the necessary information is not available in the grid information system (MDS).

The very first step consists in launching the Mico name service daemon on one of the selected compute nodes and retrieve its CORBA reference (IOR). Once the name service's IOR has been obtained, the component servers are launched on the grid resources in the same way as the name service daemon is started: the component servers may also return their IOR to the deployment tool (even if this is not mandatory, since the component servers' IORs can be retrieved using the name service). Once the component servers have been launched, the Globus Toolkit need not create any more processes: the components can be created by thread creation within the component servers, possibly several components per component server depending on the process collocation constraints expressed in the component assembly package descriptor. The component creations can be initiated from the client side (CORBA component creation requests executed in the deployment tool) or

from the server side (creation requests within the component server). We initiate remote component creations from the deployment tool (client side) to avoid modifying already existing component server programs. As the component creation gets initiated from the client side, then it is better to have the component servers send their IOR directly to the deployment tool, rather than having the deployment tool poll the name service until it holds the component servers' IORs. This is done using a simple wrapper program around the component server.

More specifically, the deployment process of the name service is exactly the same as of a component server. The deployment tool first creates a Globus GASS server which will receive the IOR of the remotely created object (name service daemon or component server) over a secured channel. The URL (hostname and port) where the GASS server is listening is retrieved by the deployment tool. Then it launches the remote process (name service daemon or component server) using GRAM remote job submission facility, passing to the remote process the URL of the GASS server among other arguments (including CORBA or MicoCCM specific arguments). The RSL script used for job submission specifies the list of files (input data, component dependencies, *etc.*) which must be staged in to the remote node prior to process execution. Using GridFTP, third-party transfers are permitted to stage in those files directly from a repository within the grid, which may not be located on the deployment client. If the process creation succeeds on the compute node selected to execute a component server or a name service daemon, the IOR of the newly created object is sent back to the deployment client using the GASS contact URL, and the GRAM job submission function in the deployment tool returns a handle, *i.e.* a Globus job ID useful to control the processes created by GRAM.

Thus the deployment tool possesses both Globus job IDs and CORBA IORs to control the processes (name service daemon and component servers) on the remote machines. For instance, an application can be stopped using the CORBA way and IORs (by calling the `ComponentServer::remove()` method) or using the Globus way and job IDs (by calling `globus-job-cancel`). Eventually, these control handles may permit a re-deployment of a component-based application or they may allow to dynamically plug visualization components to the running application.

All the data transfers using GridFTP and GASS are reliable and secured using GSI authentication and authorization mechanisms.

6 Related Work

ICENI (the Imperial College e-Science Networked Infrastructure [38, 15]) is a service oriented, integrated grid middleware which provides a component programming model to aid the application developer in constructing grid applications. The main focus of ICENI is meta-data information (performance, behavior) on component implementations in order to select and map the component implementations (corresponding to given interfaces) best-

suited to the available computing resources of a grid. The selection and mapping are constrained by user-level (pricing, quality of service) and application-level requirements [14, 13]. The ICENI project currently supports deployment using *Fork* (using the Unix secure shell `ssh`) and plans to eventually resort to the Globus Toolkit, Condor, Sun GridEngine in unspecified ways [12, 42]. Currently, the deployment using the Globus Toolkit or Condor are not implemented [17].

The *Partitionable Services Framework* (PSF) and its deployment module *Smock* assume that so-called "wrappers" have already been deployed on each node [18]; the way those wrappers get deployed and how components get uploaded to the execution servers are not specified: their experiments are conducted using an emulated network. PSF also includes *Sekitei* [19], a planner implementing an algorithm to choose component implementations, link the components together and map them to computing resources: the placement algorithm is evaluated using simulation, with no actual deployment of processes.

The *Common Component Architecture* [30, 1] (CCA) proposes a component model similar to the abstract model of CORBA. It does not provide any detail on the automatic deployment process in grid environments [3]. A recent work [16] has focused on merging the CCA component model with the OGSI framework. It is different from our work since we support both CORBA and Globus communication models.

The *Assembly and Deployment Toolkit* [24] provided by Frank Pilhofer makes it very easy to configure and create component archives and component assembly archives. However the deployment feature is limited to a *single machine* already running a MicoCCM daemon *manually launched* by the user of the toolkit, with no security enforced, so it is ill-suited for deployment in grid environments.

The *Concerto* [5] platform aims to support the deployment of parallel components [25] on *clusters of workstations*: as this project does not envisage deployment on computational grids, it does not face the problem of heterogeneity inherent to computational grids, nor does it need to enforce security (authentication, authorization, encryption) because it assumes a trusted environment and trusted users.

7 Conclusion and Future Work

The deployment of high bandwidth wide-area networks has led computational grids to offer a very powerful computing resource. In particular, this inherently distributed resource is well-suited for multiphysics applications. To face the complexity of such applications as well as the heterogeneity and volatileness of grids, the software component technology appears to be a very adequate programming model. We choose to work with the CORBA component model because its deployment model is very complete: it specifies the deployment of a set of components on a set of distributed (component) servers. However, it specifies neither how to select those nodes, nor how to create the component servers. On the other hand, a grid

access middleware, such as the Globus Toolkit, deals with security enforcement, resource, information, data management, and portability.

This paper studies the deployment of CORBA component-based applications in a computational grid using the Globus Toolkit. Its main result is that both models are complementary, not conflicting. This result seems to be generalizable to other component models and grid middleware systems. CCM does not specify how machines are to be selected, nor how component servers (processes) should be created whereas the Globus Toolkit only encompasses the start of remote processes. Moreover, grid services like data management are very useful to stage in component binaries. We have validated our analysis by actual experiments on our private computational grid. The deployment tool is currently hand-written and specific to our test case.

Our future works can be divided into three issues. The first issue is to integrate a planner for *automatically* selecting the machines on which the components will be deployed. The second issue concerns the extension of existing grid information services: while information related to compute *nodes* (CPU speed, memory size, operating system, *etc.*) is properly described in currently available resource information systems, the *network* topology and its characteristics are generally not described in a satisfactory, simple, synthetic and complete way. The last issue will be to support Grid Services. As we try not to have only one model for the component model and the grid middleware model, it should be straightforward to migrate to a web service enabled Globus Toolkit like GT4.

References

[1] Rob Armstrong, Dennis Gannon, Al Geist, Katarzyna Keahey, Scott Kohn, Lois McInnes, Steve Parker, and Brent Smolinski. Toward a common component architecture for high-performance scientific computing. In *Proc. of the 8th IEEE International Symposium on High Performance Distributed Computing (HPDC'99)*, pages 115–124, Redondo Beach, CA, August 1999.

[2] Joseph Bester, Ian Foster, Carl Kesselman, Jean Tedesco, and Steven Tuecke. GASS: A data movement and access service for wide area computing systems. In *Proc. of the 6th Workshop on Input/Output in Parallel and Distributed Systems (IOPADS)*, pages 78–88, Atlanta, GA, May 1999. ACM Press.

[3] Randall Bramley, Kenneth Chiu, Shridhar Diwan, Dennis Gannon, Madhusudhan Govindaraju, Nirmal Mukhi, Benjamin Temko, and Madhuri Yechuri. A component based services architecture for building distributed applications. In *Proc. of the 9th IEEE International Symposium on High Performance Distributed Computing (HPDC'00)*, pages 51–59, Pittsburgh, PA, August 2000.

[4] Ethan Cerami. *Web Services Essentials.* O'Reilly & Associates, 1st edition, February 2002.

[5] Luc Courtrai, Frédéric Guidec, Nicolas Le Sommer, and Yves Mahéo. Resource management for parallel adaptive components. In *Proc. of the 5th International Workshop on Java for Parallel and Distributed Computing (JPDC), held in conjunction with the 17th International Parallel and Distributed Processing Symposium (IPDPS'2003)*, pages 134–140, Nice, France, April 2003.

[6] Karl Czajkowski, Steven Fitzgerald, Ian Foster, and Carl Kesselman. Grid information services for distributed resource sharing. In *Proc. of the 10th IEEE International Symposium on High-Performance Distributed Computing (HPDC-10)*, pages 181–194, San Francisco, CA, August 2001.

[7] Karl Czajkowski, Ian Foster, Nick Karonis, Carl Kesselman, Stuart Martin, Warren Smith, and Steven Tuecke. A resource management architecture for metacomputing systems. In *Proc. of the IPPS/SPDP'98 Workshop on Job Scheduling Strategies for Parallel Processing*, volume 1459 of *LNCS*, pages 62–82, 1998.

[8] Alexandre Denis, Christian Pérez, and Thierry Priol. Towards high performance CORBA and MPI middlewares for grid computing. In Craig A. Lee, editor, *Proc. of the 2nd International Workshop on Grid Computing*, number 2242 in LNCS, pages 14–25, Denver, CO, November 2001. Springer-Verlag. held in conjunction with SuperComputing 2001 (SC'01).

[9] Ian Foster and Carl Kesselman. The Globus Project: a status report. In *Proc. of the 7th Heterogeneous Computing Workshop, held in conjunction with IPPS/SPDP'98*, pages 4–18, Orlando, FL, March 1998.

[10] Ian Foster and Carl Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*, chapter Computational Grids, pages 15–51. Morgan Kaufmann, San Francisco, CA, January 1998.

[11] Ian T. Foster, Carl Kesselman, Gene Tsudik, and Steven Tuecke. A security architecture for computational grids. In *Proc. of the 5th ACM Conference on Computer and Communications Security*, pages 83–92, San Francisco, CA, 1998. ACM Press, New York, NY.

[12] Nathalie Furmento, , William Lee, Steven Newhouse, and John Darlington. Test and deployment of ICENI, an integrated grid middleware on the UK e-Science grid. In *Proc. of the UK e-Science All Hands Meeting*, pages 192–195, Nottingham, UK, September 2003.

[13] Nathalie Furmento, Anthony Mayer, Stephen McGough, Steven Newhouse, and John Darlington. A component framework for HPC applications. In *Proc. of the 7th International Euro-Par Conference*, volume 2150 of *LNCS*, pages 540–548, Manchester, UK, August 2001.

[14] Nathalie Furmento, Anthony Mayer, Stephen McGough, Steven Newhouse, Tony Field, and John Darlington. Optimisation of component-based applications within a grid environment. In *Proc. of the 2001 ACM/IEEE conference on Supercomputing*, page 30, Denver, CO, November 2001. ACM Press, New York, NY, USA.

[15] Nathalie Furmento, Anthony Mayer, Stephen McGough, Steven Newhouse, Tony Field, and John Darlington. ICENI: Optimisation of component applications within a grid environment. *Journal of Parallel Computing*, 28(12):1753–1772, 2002.

[16] Madhusudhan Govindaraju, Sriram Krishnan, Kenneth Chiu, Aleksander Slominski, Dennis Gannon, and Randall Bramle. Merging the CCA component model with the OGSI framework. In *Proc. of the 3rd International Symposium on Cluster Computing and the Grid (CCGrid2003)*, pages 182–189, Tokyo, Japan, May 2003.

[17] ICENI Research Group. Imperial college e-Science networked infrastructure. User's guide, The London e-Science Centre, January 2004. `http://www.lesc.ic.ac.uk/iceni/downloads/guide.pdf`.

[18] Anca-Andreea Ivan, Josh Harman, Michael Allen, and Vijay Karamcheti. Partitionable services: A framework for seamlessly adapting distributed applications to heterogeneous environments. In *Proc. of the 11th IEEE International Symposium on High Performance Distributed Computing (HPDC-11)*, pages 103–112, Edinburgh, Scotland, July 2002.

[19] Tatiana Kichkaylo, Anca-Andreea Ivan, and Vijay Karamcheti. Constrained component deployment in wide-area networks using AI planning techniques. In *Proc. of the 17th International Parallel and Distributed Processing Symposium (IPDPS'2003)*, page 3, Nice, France, April 2003.

[20] Muthucumaru Maheswaran and Howard Jay Siegel. A dynamic matching and scheduling algorithm for heterogeneous computing systems. In *Proc. of the 7th Heterogeneous Computing Workshop, held in conjunction with IPPS/SPDP'98*, pages 57–69, Orlando, FL, March 1998.

[21] Object Management Group (OMG). Deployment and configuration of component-based distributed applications specification. Draft Adopted Specification ptc/03-07-02, `http://www.OMG.org/cgi-bin/apps/doc?ptc/03-07-02.pdf`, June 2003.

[22] Open Management Group (OMG). CORBA components, version 3. Document formal/02-06-65, June 2002.

[23] Open Management Group (OMG). Common Object Request Broker Architecture (CORBA/IIOP). Document formal/02-11-03, November 2003.

[24] Frank Pilhofer. Assembly and deployment toolkit. `http://www.fpx.de/MicoCCM/Toolkit/`.

[25] Christian Pérez, Thierry Priol, and André Ribes. A parallel CORBA component model for numerical code coupling. *The International Journal of High Performance Computing Applications (IJHPCA)*, 17(4):417–429, 2003. Special issue Best Applications Papers from the 3rd International Workshop on Grid Computing.

[26] Resource Specification Language (RSL) version 1.0. `http://www-fp.globus.org/gram/rsl_spec1.html`.

[27] Sun Microsystems. Enterprise JavaBeans Specification, August 2001.

[28] Martin Swany and Rich Wolski. Representing dynamic performance information in grid environments with the network weather service. In *Proc. of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid'02)*, pages 48–56, Berlin, Germany, May 2002.

[29] Clemens Szyperski. *Component Software: Beyond Object-Oriented Programming.* Addison-Wesley / ACM Press, first edition, 1998.

[30] The CCA Forum web site. `http://www.CCA-forum.org/`.

[31] The Common TeraGrid Software Stack (CTSS). `http://www.TeraGrid.org/userinfo/guide_software.html`.

[32] The DataGrid Project web site. `http://www.eu-datagrid.org/`.

[33] The DOE Science Grid web site. `http://DOEScienceGrid.org/`.

[34] The Globus Alliance web site. `http://www.Globus.org/`.

[35] The Globus Project. GridFTP: Universal data transfer for the grid. White paper, `http://www.globus.org/datagrid/deliverables/C2WPdraft3.pdf`, September 2000.

[36] The Globus Project. GridFTP update. Technical report, `http://www.globus.org/datagrid/deliverables/GridFTP-Overview-200201.pdf`, January 2002.

[37] The Grid Physics Network (GriPhyN) web site. `http://www.GriPhyN.org/`.

[38] The ICENI web site at the London e-Science Centre. `http://www.lesc.ic.ac.uk/iceni/`.

[39] The Object Management Group (OMG) web site. `http://www.OMG.org/`.

[40] The TeraGrid web site. `http://www.TeraGrid.org/`.

[41] Richard Wolski. Forecasting network performance to support dynamic scheduling using the network weather service. In *Proc. of the 6th International Symposium on High-Performance Distributed Computing (HPDC-6'97)*, pages 316–325, Portland, OR, August 1997.

[42] Laurie Young, Stephen McGough, Steven Newhouse, and John Darlington. Scheduling architecture and algorithms within the ICENI grid middleware. In *Proc. of the UK e-Science All Hands Meeting*, pages 5–12, Nottingham, UK, September 2003.

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399