Towards a common deployment model for Grid systems

Massimo Coppola[1], Marco Danelutto[2], Sébastien Lacour[3], Christian Pérez[3], Thierry Priol[3], Nicola Tonellotto[1,4], and Corrado Zoccolo[2]

[1] Information Science and Technologies Institute, Italian National Res. Council, Pisa
[2] Dipartimento di Informatica, Università di Pisa, L.go B. Pontecorvo, 3, 56127 Pisa, Italy
[3] IRISA/INRIA, Campus de Beaulieu, 35042 Rennes Cedex, France
[4] Information Engineering Department, Università di Pisa, Italy

Abstract. Deploying applications within a Grid infrastructure is an important aspect that has not yet be fully addressed. This is particular true when it is offered to the programmers some high-level abstractions. Supporting such high-level abstractions, like objects or components, requires the deployment of several middleware systems that correspond to the implementation of such high-level of abstractions. The deployment problems is thus not anymore related to the application itself but encompass a more general level of concern about how to describe which middleware systems are required for a given application, how resources are discovered and selected according to the application requirement, how the deployment process is planned and enacted and finally how the application is executed. This paper addresses this deployment issue by illustraing how it has been handled within two projects (ASSIST and GridCCM). From this experience, a common deployment model is presented as the result of the integration of the experience gained by researchers involved in these two projects.

1 Introduction

The Grid vision introduced in the end of the nineties has now become a reality with the availability of quite a few Grid infrastructures, most of them experimental but some others will come soon in production. Although most of the research and development efforts have been spent in the design of Grid middleware systems (i.e., the Grid operating systems), the question of how to program such large scale computing infrastructures remains open. It is no doubt that programming such computing infrastructures will be extremely complex considering its parallel and distributed nature. The programmer vision of a Grid infrastructure is often determined by its programming model. The level of abstraction that is proposed today is rather low, giving the vision either of a parallel machine, with a message-passing layer such as MPI, or a distributed system with a set of services, such as Web Services, to be orchestrated. Both of these two approaches offer a very low level programming abstraction and are not really adequate, limiting the spectrum of applications that could take

benefit from Grid infrastructures. To overcome this situation, it is of paramount importance to propose high level abstractions to facilitate the programming of Grid infrastructures and in a longer term to be able to develop more secure and robust next generation Grid middleware systems by using these high level abstractions for their design. Thus, the current situation is very similar to what happened with computers in the sixties: minimalist operating systems were developed first with assembly languages before being developed in the seventies by languages that offer higher levels of abstraction.

Several research groups are already investigating how to design or adapt programming models that provide this required level of abstraction. Among these models, object-oriented or component-oriented programming models are good candidates to deal with the complexity of programming Grid infrastructures. A Grid application can be seen as a collection of components interconnected in a certain way that must be deployed on available computing resources managed by the Grid infrastructure. Components can be reused for new Grid applications, reducing the time to build new applications. However, from our experience such models have to be combined with other programming models that are required within a Grid infrastructure. It is imaginable that a parallel program can be encapsulated within a component. Such a parallel program is based on a parallel programming model which might be for instance message-based or skeleton-based. Moreover, object or component oriented programming can be coupled with a service oriented approach exposing some object methods or component ports as services through the use of Web Services.

The results of this is that this combination of several models to design Grid applications lead to a major challenge: the deployment of applications within a Grid infrastructure. Such programming models are always implemented through various runtime or middleware systems that have their own dependencies vis-à-vis of operating sytems, making it extremely challenging to deploy applications within a heterogeneous environment, which is an intrinsic property of a Grid infrastructure.

The objective of this paper is to propose a common deployment model based on the experience gained from the ASSIST and GridCCM projects. This paper is organized as follows. Section 2 gives a short description of the ASSIST and GridCCM projects in which solutions to the deployment problem have been given. Section 3 describes our common analysis of what should be the different steps to deploy grid applications. Section 4 gives a short description of the two deployments systems we have designed for ASSIST and GridCCM. Finally, Section 5 concludes the paper and presents some perspectives.

2 Deploying software components with ASSIST and GridCCM

Both University of Pisa and INRIA-Rennes have investigated the problem of deploying component-based Grid applications in the context of the ASSIST

and GridCCM programming environments and came out with two approaches with some similarities and differencies. In the framework of the CoreGRID Network of Excellence, the two research groups decided to join their efforts to develop a common deployment model suitable for both projects taking benefits of the experience of both groups. The remaining part of this section, the ASSIST and GridCCM programming and component models are presented, so as to illustrate the common requirements on the deployment system.

2.1 Assist

ASSIST (A Software development System based upon Integrated Skeleton Technology) is a complete programming environment [**?**] aimed at the efficient development (w.r.t. development complexity and overall performance) of high-performance multi-disciplinary applications, at managing their complexity and lowering their time to market. To achieve this target in a Grid Computing settings, the environment must deal with heterogeneous resources, and with dynamic changes in the computational behaviour of resources and application modules.

ASSIST provides an efficient and high-performance way of expressing the parallel computational cores, and the Grid.it component [**?**] support allows for 1) integration of different component frameworks, 2) implementation of automatic component adaptation to varying resource performance or varying computational needs of the program.

The deployment of an ASSIST application or Grid.it component is a complex task, involving the selection of resources and configuration of diffenent set of processes in such a way that they are able to cooperate, obtaining high-performances. Moreover, this task does not finish when an application is launched: an application component can, in fact, request more resources, in order to adapt [**?**] its configuration to fulfill specified performance requirements.

ASSIST components have a platform independent description encoded in ALDL [**?**], an XML dialect expressing the detailed requirements and execution constraints of the elementary blocks composing the component.

Within the ASSIST environment, the Grid Execution Agent (GEA) module takes in charge the ALDL description of each component and interacts with different Grid middlewares. GEA selects resources, maps processes, deploys them. Moreover, GEA reiterates previous steps to perform resource allocation changes (dynamically adaptive behaviour).

2.2 GridCCM: a parallel component model

GridCCM [**?**] is a research prototype that targets scientific code coupling applications. Its programming model extends the CORBA Component Model (CCM) with the concept of parallel components. A parallel component, defined as a collection of sequential components, aims at embedding a parallel code whatever the parallelism technology is (MPI, PVM, OpenMP, ...). Interaction between parallel components are handled by GridCCM which supports optimized

scheduled MxN communications. MxN data redistributions and communication scheduling may be extended at user-level.

The deployment of a GridCCM application turns out to be a complex task because several middleware systems may be involved. There are the component middleware, which implies to deploy CCM applications, and the technology used by the parallel component which may be MPI, PVM or OpenMP for example. Moreover, to deal with network issues, an environment like PadicoTM [**?**] should be also deployed with the application.

The description of an GridCCM application is achieved thanks to an extension of the XML CCM Component Software Description language. Hence, a parallel component has a parallel implementation whose description may be a MPI description [**?**].

To handle the complexity of deploying such an application, we have conceived a deployment tool named ADAGE [**?**]. While the installation and configuration of CCM application are already tackled, an originality of ADAGE stems in the deployment of CCM application into a Grid. Another originality is its support of multi-middleware application like GridCCM. Hence, it is able to deal with GridCCM applications which may currently mix CCM and MPI descriptions.

2.3 Discussion

At the light of these works, it appears that a common understanding of the deployment process can be build from our experience. Both ASSIST and GridCCM expose programming models that required advanced deployment tools to efficiently handle the different elements of an application to be deployed. Moreover, they provide distinct features like the dynamic behavior and the different Grid middleware support of ASSIST and the multi-middleware application support of GridCCM.

3 General Overview of the Deployment Process

Starting from a description of an application and a user objective function, the deployment process is responsible for automatically performing all the steps needed to start the execution of the application on a set of selected resources. This is done in order to avoid the user from directly dealing with heterogeneous resource management mechanisms.

From the point of view of the execution, a Grid component contains a structured set of binary executables and requirements for their instantiation. Our objectives include generating deployment plans

- to deploy components in a multi-middleware environment,
- to dynamically alter a previous configuration, adding new computational resources to a running application,
- for re-deployment, when a complete restart from a previous checkpoint is needed (severe performance degradation or failure of several resources).

A framework for the automatic execution of applications can be composed of several interacting entities in charge of distinct activities, as depicted in Fig. 1; these activities are not necessarily executed in a predetermined order, and one or more of them can be executed more than once, depending on their implementation and on the characteristics of the application and the Grid.

Fig. 1. Activities involved in the deployment process of an application

In the following we illustrate in detail the activities involved in the deployment of an application on a set of resources, as well as the input that must be provided by the user or the deployment framework to perform such activities.

3.1 Application Submission

This is the only activity which the user must be involved in, to provide the information necessary to drive the following phases. This information is provided through a file containing a description of the components of the application, of their interactions, and of the required resource characteristics.

Application Description. The description of the submitted application, written in an user-understandable specification language, is composed of:

- **module description**: the executable files, I/O data and configuration files which make up each application module;
- the information to guide the stages related to application mapping onto resources:
  - **resource constraints**: the characteristics that Grid resources (computational, storage and network) must possess to execute the application,
  - **execution platform constraints**: the software (libraries, middleware systems) that must be installed to satisfy application dependencies;
  - **placement policies**: restrictions or hints for the placement of subsets of application processes (e.g. co-location, location within a specific network domain, or network performance requirements, etc.);

- **resource ranking**: an objective function provided by the user, stating the optimization goal of application mapping. Resource ranking is exploited to select the best resource, or set of them, among those satisfying the given requirements for a single application process;

Resource constraints can be expressed as *unitary requirements*, that is requirements that must be respected by a single module or resource (e.g. CPU rating), and *aggregate requirements*, i.e., requirements that a set of resources or a module group must respect at the same time (e.g. all the resources on the same LAN, access to a shared filesystem); some placement policies are implicitly aggregate requirements.

– **deployment directives**: determine the tasks that must be performed to set up the application runtime environment and to start the actual execution.

The provided information is used throughout the deployment process: the evidence will be provided in the following sections.

3.2 Resource Discovery

This activity is aimed at finding the resources that can host the execution of the application. In the application description several requirements can be specified that available resources must respect to be eligible for execution. The requirements can specify hardware characteristics (e.g., CPU rating, available memory, disk space, free TCP sockets), software available on the resource (e.g. OS, libraries, compilers, runtime environments), services needed to deploy components (e.g., ability to transfer files using particular protocols, to execute scripts), services needed to execute the hosting environment of the application (e.g., configuration of installed runtime environments).

Resources fulfilling unitary requirements can be discovered, interacting with Grid Information Services. Then, the information needed to perform resource selection (that considers also aggregate requirements), must be collected, for each suitable resource found.

The GIS can be composed of various software systems, implementing information providers that communicate with different protocols (MDS-2, MDS-3, MDS-4, NWS, iGrid, custom). Some of these systems provide only static information, while others can report dynamic information about resource state and performance, including network topology and available bandwidth. In order to interact with such different entities, an intermediate translation layer between the requirements needed by the user and the information provided may be necessary.

3.3 Resource Selection

When information about available resources is collected, the proper resources that will host the execution of the application must be selected. This activity also implies satisfying all the aggregate requirements within the application. Thus,

It can require repeated interaction with the resource discovery mechanisms to find the best set of resources, and it can exploit dynamic information.

At this point, the user objective function must be evaluated against the characteristics and available services of the resources, establishing a resource ranking where appropriate in order to find a suitable solution.

3.4 Deployment Planning

A component-based application can require different services installed on the selected resources to host its execution. Moreover, additional services can be transferred/activated on the resources or configured to set up the hosting environment.

Each kind of these ancillary applications has a well-defined deployment schema, that describes the workflow of actions needed to set up the hosting environment berfore the actual execution can start.

After resource selection, an abstract deployment plan is devised by gathering the deployment schemata of all application modules. The abstract plan is then mapped on the resources, and turned into a concrete plan, identifying all the services and protocols that will be exploited in the next phase on each resource, in order to set up and start the runtime environment of the application.

For example, if files must be transferred, a transfer protocol (e.g. HTTP, FTP, GridFTP, SCP) must be selected, the services must be configured/started (e.g. custom HTTP server, client FTP service) on some of the resources, and environment variables must be declared. At the end of this phase, the concrete deployment plan must be generated, specifying every single task to perform to deploy the application.

This activity can require repeated interactions with the resource discovery and selection phases because some problems about the transformation from the deployment schema to the deployment plan can arise, thus the elimination of one or more eligible resources can force to find new resources and restart the whole planning process.

3.5 Deployment Enactment

The concrete deployment plan developed in the previous phase is submitted to the execution framework, which is in charge of the execution of the tasks needed to deploy the application. This service must ensure the right execution of the deployment tasks while respecting the time dependencies between them. At the end of this phase, the execution environment of the application must be ready to start the actual execution.

This activity must deal with different kind of softwares and middlewares; the selection of the right one depends on the concrete deployment plan. The implementation of the services that will perform this activity must be flexible enough to implement the functionalities to interact with different services, as well as to add mechanisms to deal with new services.

Changes of the state of the resources can force a new deployment plan of some tasks, so this phase can require interactions with the previous one.

3.6 Application Execution

The deployment process for adaptive Grid applications does not end when the application is started. Several activities have to be performed while the application is already active, and actually suggest that the deployment system must rely on at least one permanent process or daemon. The whole application lifecycle must be managed, in order to support new resource requests for application adaptation, to schedule a restart if application failure is detected, and to release resources when normal termination is reached. These monitoring and controlling activities have to be mediated by the deployment support (actual mechanisms depend on the middleware), and they even cannot be reliably performed over noisy/low-bandwidth/mobile networks.

Finally, we want to underline that when altering the current deployment (or redeploying from scratch), the application execution phase interacts with and can possibly repeat all previous deployment phases.

4 Current prototypes

4.1 GEA

The Grid Execution Agent (GEA, [**?**]) is a automatic tool developed in the *Grid.it* Project to seamlessly run complex component-based Grid applications on different middlewares. The application classes targeted are mainly high-performance, data and/or computation intensive, and distributed ones.
GEA is a open framework implementing several interfaces; these interfaces provide the functionalities needed to effectively deploy on a Grid complex applications, developed with high-level programming tools like **ASSIST**. GEA, starting from a description of the application and its resource requirements, automatically performs resource discovery and selection, handles data and executable file transfers. Furthermore it plans and enacts the deployment workflow of **ASSIST** components and starts the execution of the their processes, monitoring their completion.
In detail, the Grid Execution Agent provides the following capabilities:

- Virtualization of several basic functions w.r.t. the underlying middleware (e.g. SSH, Globus, ...); we call this level the Grid Abstract Machine (GAM) [**?**,**?**] level. The GAM level provides an abstraction of security mechanisms, resource discovery, resource selection, (secure) code & data staging and launch of the executables.
- Resource location and monitoring during the whole deployment and execution phases.
- Instantiation, monitor and termination of **ASSIST** components w.r.t. its workflow deployment model.

– Automatic reconfiguration of components when the resources in the Grid change state significantly.

In GEA, we assume that the initial description of the application is given in a general format (ALDL [**?**] in our case), which can encode the information needed in all the sub-cases. As a result of the deployment process, different parts of the ALDL description are translated into the appropriate forms for the middleware used on each part of the application.

4.2 ADAGE

ADAGE [**?**] (*Automatic Deployment of Applications in a Grid Environment*) is a research project that aims at studing the deployment issues related to multi-middleware applications. One of its originality is to use a generic application description model (GADe) [**?**] to be able to handle several middleware systems. ADAGE follows the deployment process described in this paper for the features it supports.

With respect to application submission, ADAGE requires a application description, which is specific to a programming model, a reference to a resource information service (MDS2, or an XML file), and a control parameter file. The application description is internally translated into a generic description so as to support in particular multi-middleware applications. The control parameter file allows a user to express constraints on the placement policies which are specific to an execution. For example, a constraint may affect the latency and bandwidth between a computational component and a visualization component.

The support of multi-middleware application is based on a plugin mechanism. The plugin is involved in the conversion from the specific to the generic application description but also during the configuration phase.

ADAGE currently deploys only static applications. It supports standards programming model like MPI (MPICH1-P4 and MPICH-G2), CCM or JXTA and also more advanced programming models like GridCCM. The current supports of GridCCM is restricted to MPI-based parallel components.

4.3 Comparison of GEA and ADAGE

Table 1 sumps up the similarites and difference between GEA and ADAGE with respect to the features of our common deployment process.

5 Conclusion

References

1. M. Aldinucci, S. Campa, M. Coppola, M. Danelutto, D. Laforenza, D. Puppin, L. Scarponi, M. Vanneschi, and C. Zoccolo. Components for high performance Grid

Table 1. Features of the common deployment process supported by GEA and ADAGE.

Feature	GEA	ADAGE
Component description in input	ALDL (generic)	Many, via GADe[a]
Multi-middleware application	Yes	Yes
Dynamic application	Yes	No
Resource constraints	Yes	Yes
Execution constraints	Yes	Yes
Grid Middleware	Many, via GAM[b]	ssh and Globus2

[a] MPI, CCM, GridCCM, JXTA,...
[b] Globus toolkit 2,3,4, ssh, etc.

programming in the Grid.it project. In V. Getov and T. Kielmann, editors, *Proc. of the Workshop on Component Models and Systems for Grid Applications (June 2004, Saint Malo, France)*. Springer, January 2005.

2. M. Aldinucci, M. Coppola, M. Danelutto, M. Vanneschi, and C. Zoccolo. ASSIST as a research framework for high-performance Grid programming environments. In J. C. Cunha and O. F. Rana, editors, *Grid Computing: Software environments and Tools*. Springer, 2005. (to appear, draft available as TR-04-09, Dept. of Computer Science, University of Pisa, Italy, Feb. 2004).
3. M. Aldinucci, A. Petrocelli, E. Pistoletti, M. Torquati, M. Vanneschi, L. Veraldi, and C. Zoccolo. Dynamic reconfiguration of grid-aware applications in ASSIST. In *11th Intl Euro-Par 2005: Parallel and Distributed Computing*, LNCS, Lisboa, Portugal, August 2005. Springer. To appear. Draft available as Technical Report TR-05-05, Computer Science Dept, University of Pisa.
4. Marco Danelutto, Marco Vanneschi, Corrado Zoccolo, Nicola Tonellotto, Ranieri Baraglia, Tiziano Fagni, Domenico Laforenza, and Alessandro Paccosi. Hpc application execution on grids. Dagstuhl seminar 04451 – Future Generation Grids 2004, November 2004. (`http://www.dagstuhl.de/04451/materials/`).
5. Alexandre Denis, Christian Pérez, and Thierry Priol. Padicotm: An open integration framework for communication middleware and runtimes. *Future Generation Computer Systems*, 19(4):575–585, May 2003.
6. Sébastien Lacour, Christian Pérez, and Thierry Priol. A software architecture for automatic deployment of CORBA components using grid technologies. In *Proceedings of the 1st Francophone Conference On Software Deployment and (Re)Configuration (DECOR'2004)*, pages 187–192, Grenoble, France, October 2004.
7. Sébastien Lacour, Christian Pérez, and Thierry Priol. Description and packaging of MPI applications for automatic deployment on computational grids. Research Report RR-5582, INRIA, IRISA, Rennes, France, May 2005.
8. Sébastien Lacour, Christian Pérez, and Thierry Priol. Generic application description model: Toward automatic deployment of applications on computational grids. In *6th IEEE/ACM International Workshop on Grid Computing (Grid2005)*, Seattle, WA, USA, November 2005. Springer-Verlag.
9. Christian Pérez, Thierry Priol, and André Ribes. A parallel CORBA component model for numerical code coupling. *The International Journal of High Performance Computing Applications*, 17(4):417–429, 2003.

10. M. Vanneschi. The programming model of ASSIST, an environment for parallel and distributed portable applications. *Parallel Computing*, 28(12):1709–1732, December 2002.