TOWARDS A COMMON DEPLOYMENT MODEL FOR GRID SYSTEMS

Massimo Coppola and Nicola Tonellotto
ISTI
Area della Ricerca CNR, 56124 Pisa
Italy
coppola@di.unipi.it
nicola.tonellotto@isti.cnr.it

Marco Danelutto and Corrado Zoccolo
Dept. of Computer Science, University of Pisa
L.go B. Pontecorvo, 3, 56127 Pisa
Italy
marcod@di.unipi.it
zoccolo@di.unipi.it

Sébastien Lacour and Christian Pérez and Thierry Priol
IRISA/INRIA
Campus de Beaulieu, 35042 Rennes Cedex
France
Christian.Perez@irisa.fr
Thierry.Priol@irisa.fr

Abstract Deploying applications within a Grid infrastructure is an important aspect that has not yet been fully addressed. This is particularly true when high-level abstractions, like objects or components, are offered to the programmers. High-level applications are built on run-time supports that require the deployment process to span over and coordinate several middleware systems, in an application independent way. This paper addresses deployment by illustrating how it has been handled within two projects (ASSIST and GridCCM). As the result of the integration of the experience gained by researchers involved in these two projects, a common deployment process is presented.

Keywords: Grid computing, deployment, generic model.

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, the question of how to program such large scale computing infrastructures remains open. Programming such computing infrastructures will be quite 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 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. Of course such approaches may be sufficient for simple applications but a Grid infrastructure has to be generic enough to also handle complex applications with ease. To overcome this situation, it is required 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 as well. 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, 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, a component oriented programming model can be coupled with a service oriented approach exposing some 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 leads 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 systems, 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 process based on the experience gained from the ASSIST and GridCCM projects. This paper is organized as follows. Section 2 gives an overview of the ASSIST and GridCCM projects. Section 3 presents our common analysis of what should be the different steps to deploy grid applications. Section 4 shortly describes GEA and Adage, the two deployment systems designed respectively for ASSIST and GridCCM, and how they already conform to the common model. Finally, Section 5 concludes the paper and presents some perspectives.

2. ASSIST and GridCCM Software Component Models

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 differences. In the framework of the CoreGRID Network of Excellence, the two research groups decided to join their efforts to develop a common deployment process suitable for both projects taking benefits of the experience of both groups. In 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 [13]) is a complete programming environment aimed at efficient development of high-performance multi-disciplinary applications. Efficiency is pursued both w.r.t. the development effort and to overall performance, as the ASSIST approach aims at managing the complexity of applications, easing prototyping and decreasing time-to-market.

ASSIST provides a basic modularization of parallel applications by means of sequential and parallel modules (*parmods*), with well-defined interfaces exploiting stream-based communications. The ASSISTcl coordination language describes modules and composition of them.

Sequential modules wrap code written in several languages (e.g. C, C++, FORTRAN). Parmods describe parallel execution of a number of sequential functions within *Virtual Processes* (VPs), mainly activated by stream communications, and possibly exploiting shared state and/or explicit synchronization at the parmod level. The abilities to (1) describe both task and data-parallel

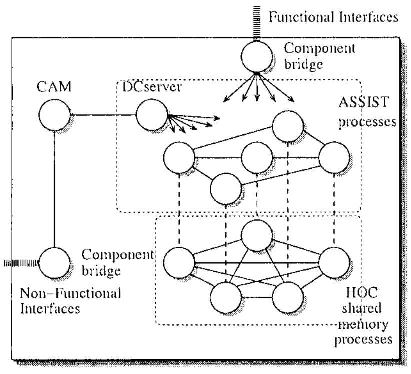

Figure 1. The process schema of a simple Grid.it component.

```
<aldl:application xmlns = ... >
  <aldl:requirement name = "libraries">
    <ns1:lib fileName = "libACE.so.5.4.0" fileSystemName =
      "/tmp" arch = "i686" executable = "no">
      <ns1:source url = "aar:///Modules/lib/libACE.so.5.4.0"/>
    </ns1:lib>                ...
  </aldl:requirement>         ...
  <aldl:requirement name = "ND001__Ivp">
    <ns1:executable master = "no" strategy = "no" arch = "i686">
      Modules/bin/i686-pc-linux-gnu/ND001__ivp    </ns1:executable>
    <ns1:hoc nHocTot = "3" nHoc = "1" prefixAlias = "ND001__shared__"
      hocExName = "hoc" hocConfName = "hoc.conf" bridge = "no"
      fileSystemName = "/tmp" arch = "all">
      <ns1:source urlExHoc = "aar:///Modules/bin/hoc" urlConfHoc
        = "aar:///Modules/svc/hoc.conf"/>
    </ns1:hoc>
  </aldl:requirement>         ...
  <aldl:requirement name = "CAM_sConfiguration">
    <ns1:executable master = "no" strategy = "yes" arch =
      "i686">Modules/bin/i686-pc-linux-gnu/CAM_s
    </ns1:executable>         ...
  </aldl:requirement>         ...
</aldl:application>
```

Figure 2. Excerpt from the ALDL describing a Grid.it component (ellipsis shown as . . .).

behavior within a parmod, (2) to fine-control nondeterminism when dealing with multiple communication channels and (3) to compose sequential and parallel modules into arbitrary graphs, they allow expressing parallel semantics and structure in a high-level, structured way. ASSIST implements program adaptivity to changing resource allocation exploiting the VP granularity as a user-provided definition of elementary computation.

ASSIST supports component-based development of software by allowing modules and graph of modules to be compiled into Grid.it components [2], and separately deployed on parallel and Grid computing platforms. The Grid.it framework supports integration with different frameworks (e.g. CCM, Web Services), and implementation of automatic component adaptation to varying resource/program behavior. Component-based applications can exploit both ASSIST native adaptivity and Grid.it higher-level "super-components", which arrange other components into basic parallel patterns and provide dynamic management of graphs of of components within an application.

ASSIST applications and Grid.it components have a platform-independent description encoded in ALDL [4], an XML dialect expressing the structure, the detailed requirements and the execution constraints of all the elementary composing blocks. Support processes shown in Fig. 1 are all described in the ALDL syntax of Fig. 2, e.g. besides those actually performing computation and implementing virtual shared memory support, we include those providing inter-component communications and interfacing to other component frameworks. ALDL is interpreted by the GEA tool (see Section 4.1), which translates requirements into specific actions whenever a new instance of a component has to be executed, or an existing instance dynamically requires new computing resources.

Summing up, the support of the ASSIST/Grid.it environment must deal with (1) heterogeneous resources, (2) dynamically changing availability and allocation of resources, (3) several sets of processes implementing application and components, which need (4) different execution protocols and information (e.g. setting up a shared memory space support versus instantiating a CORBA name service).

Deploying an application is therefore a complex process which takes into account program structure and resource characteristics, involves selecting resources and configuring several sets of processes to cooperate and obtain high-performance. Finally, the deployment task continues during program execution, processing resource requests from components, which utilize run-time reconfiguration to adapt and fulfill specified performance requirements [4].

The GEA tools has to provide these functionalities, shielding the actual application run-time support from the details of the different middleware used to manage the available resources.

2.2 GridCCM: a Parallel Component Model

The model GridCCM [12] 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. CCM specifies several models for the definition, the implementation, the packaging and the deployment of distributed components [11]. However, the embedding of a parallel code, such as an MPI-based code, into a CCM component results in a serialization of the communications with another component also embedding a parallel code. Such a bottleneck is removed with GridCCM which enables MxN communications between parallel components.

A parallel component is a component whose implementation is parallel. Typically, it is a SPMD code which can be based on any kind of parallel technology (MPI, PVM, OpenMP, ...). The only requirements of GridCCM that the distributions of input and output data need to be specified. Such distributed data can be the parameters of interface operations or can be the type of event streams. Interactions between parallel components are handled by GridCCM which supports optimized scheduled MxN communications. It is a two phase process. First, data redistribution libraries compute the communication matrices for all the distributed data of an operation invocation. These data redistributions are a priori distinct. Second, a scheduling library takes care of globally optimizing the transfer of all the data associated to the operation invocation with respect to the properties of the network like the latency, the networking card bandwidth and the backbone bandwidth for wide area networks. Data redistributions and communication scheduling libraries may be extended at user-level.

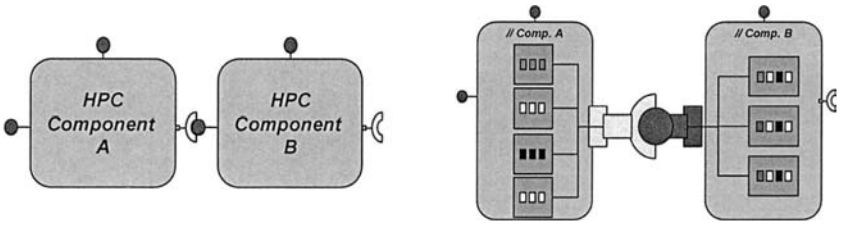

Figure 3. On the left, a parallel component appears as a standard component. On the right, communications between two parallel components are of type MxN.

```
<softpkg ...>
...
</implementation>
 <GridCCM type="MPI" id="pi1">
  <functional_prgrm>
   <location>
    http://g5k.org/Flow.mpi
   </location>
  </functional_prgrm>
  ...
 </GridCCM>
</softpkg>
```

Figure 4. Example of a GridCCM description of a MPI-based parallel component.

```
<MPI_application>
 <programs>
  <program id="master_program">
   <binary vendor="MPICH">
    <location>URL...</location>
   </binary>
  </program>
  <application>
   <world_size>32</world_size>
   ...
  </application>
</MPI_application>
```

Figure 5. Partial view of the description of the MPI-based parallel component implementation.

As illustrated in Figure 3, a parallel component looks like any CCM component and can be connected with any other CCM components. Hence, an application may be incrementally parallelized, one component after the other.

The deployment 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 [5] 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 (CSD) language. As shown in Figure 4, this extension enables the CSD to refer to another file to actually describe the structure of the parallel component implementation as displayed in Figure 5. This solution has been selected because GridCCM does not enforce any parallel technology. More information is provided in [9]. Then, Adage, a deployment tool described in Section 4.2 is used to deploy it.

As GridCCM is an extension of CCM, it implicitly provides the same heterogeneity support than CORBA for operating system, processor, compiler, libraries dependencies, etc.

2.3 Discussion

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. Hence, a common deployment process will help in integrating features needed for their deployment.

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 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 Figure 6. The logical order of the activities is fixed (Submission, Discovery, Selection, Planning, Enactment, Execution). Some steps have to be re-executed when the application configuration is changed at run-time. Moreover, the steps in the grey box, that interact closely, can be iterated until a suitable set of resources is found.

In the following we describe the activities involved in the deployment of an application on a Grid. We also detail the inputs that must be provided by the user or the deployment framework to perform such activities.

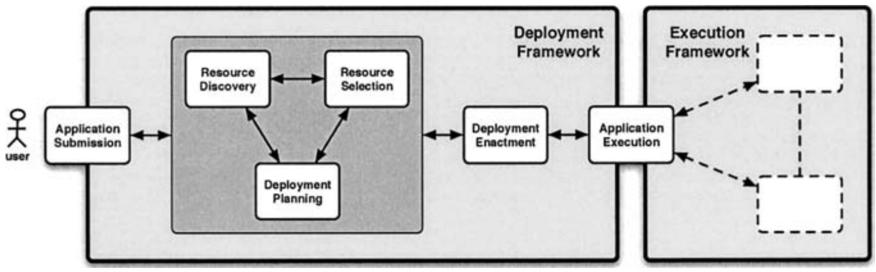

Figure 6. Activities involved in the deployment process of an application.

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.

3.1.1 Application Description.

The description of (the components of) the submitted application, written in an user-understandable specification language, is composed of various kinds of data. First, the **module description** deals with the executable files, I/O data and configuration files which make up each module (e.g. each process). Second, there is information to guide the stages related to mapping the application onto resources, like the **resource constraints** – characteristics that Grid resources (computational, storage, network) must possess to execute the application, the **execution platform constraints** – software (libraries, middleware systems) that must be installed to satisfy application dependencies, the **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), and the **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 as *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 file system); some placement policies are implicitly aggregate requirements. Third, the **Deployment directives** determine the tasks that must be performed to set up the application runtime environment, and to start the actual execution.

As discussed in the following sections, the provided information is used throughout the deployment process.

3.2 Resource Discovery

This activity is aimed at finding the resources compatible with 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), software ones (e.g. OS, libraries, compilers, runtime environments), services needed to deploy components (e.g. accessible TCP ports, specific file transfer protocols), and particular execution services (e.g. to configure the application execution environment).

Resources satisfying 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[1] 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 characteristics. In order to interact with such different entities, an intermediate translation layer between the requirements needed by the user and the information provided is necessary. Information retrieved from different sources is mapped to a standard schema for resource description that can be exploited in the following activities independently from the information source.

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, and the different parts of each component have to be mapped on some of the selected resources. This activity also implies satisfying all the aggregate requirements within the application. Thus, repeated interaction with the resource discovery mechanisms may be needed to find the best set of resources, also exploiting dynamic information.

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

[1]Grid Information Service

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

After resource selection, an abstract deployment plan is computed 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, to transfer files we must select a protocol (e.g. HTTP, GridFTP), start or configure the related services and resources, and finally start the transfer. 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 a correct execution of the deployment tasks while respecting the precedences described in the deployment plan. 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 kinds of software and middleware systems; the selection of the right ones 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 in the state of the resources can force a new deployment plan for some tasks. Hence, this phase can require interactions with the previous one.

3.6 Application Execution

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

4. Current Prototypes

4.1 GEA

In the ASSIST/Grid.it architecture the Grid Abstract Machine (GAM, [2]) is a software level providing the abstractions of security mechanisms, resource discovery, resource selection, (secure) data and code staging and execution. The Grid Execution Agent (GEA, [4]) is the tool to run complex component-based Grid applications, and actually implements part of the GAM. GEA provides virtualization of all the basic functions of deployment w.r.t. the underlying middleware systems (see Tab. 1), translating the abstract specification of deployment actions into executable actions. We outlined GEA's requirements in Sect. 2.1. In order to implement them, GEA has been designed as an open framework with several interfaces. To simplify and make fully portable its implementation, GEA has been written in Java.

As mentioned, GEA takes in charge the ALDL description of each component (Fig. 2) and performs the general deployment process outlined in Sect. 3, interacting with Grid middleware systems as needed. GEA accepts commands through a general purpose interface which can have multiple protocol adaptors (e.g. command-line, HTTP, SSL, Web Service). The first command transfers to the execution agent a compact archival form of the component code, also containing its ALDL description. The ALDL specification is parsed and associated to a specific session code for subsequent commands (GEA supports deploying multiple components concurrently, participating in a same as well as in different applications). Component information is retained within GEA, as the full set of GEA commands accepted by the front-end provides control over the life cycle of a component, including the ability to change its resource allocation (an API is provided to the application runtime to dynamically request new resources) and to create multiple instances of it (this also allows higher-level components to dynamically replicate hosted ones).

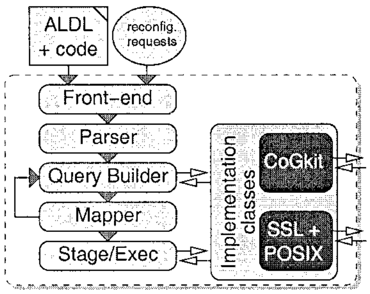

Figure 7. Overall architecture of GEA.

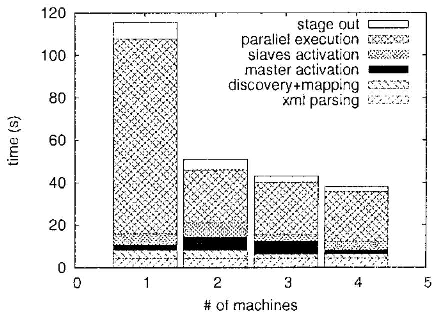

Figure 8. GEA launch time of a program over 1–4 nodes in a Globus network.

Each deployment phase described in Sect. 3 corresponds to an implementation class performing that step (see Fig. 7). GEA selects resources, maps application processes onto them, possibly loops back to the research, and finally deploys the processes, handling code and data staging in and out. This tasks are carried on according to the specific design of the class implementing each step, so that we can choose among several mapping and resource selection strategies when needed. In particular, different subclasses are available in the GEA source that handle the different middleware systems and protocols available to perform the deployment.

Current GEA architecture contains classes from the CoGKit to exploit resource location (answering resource queries through Globus MDS), monitoring (through NWS), and resource access on Globus grids. Test results deploying over 1 to 4 nodes in a local network are shown in Fig. 8. GEA also provides classes to gather resource description on clusters and local networks (statically described in XML) and to access them (assuming centralized authentication in this case). Experiments have also been performed with additional modules interfacing to a bandwidth allocation system over an optical network [14].

Different kinds of handshake among the executed processes happen in the general case (e.g. servers or naming services may need to be deployed before other application processes), thus creating a graph of dependencies among the deployment actions. This is especially important whenever a Grid.it component needs to wrap, or interact with, a CCM component or a Web Service. Currently, GEA manages processes belonging to different middleware systems within a component according to the Grid.it component deployment workflow. Work is ongoing to redesign those classes managing execution order and configuration dependencies for the "server" and "slave" processes. This will allow to parameterize the deployment workflow and to fully support different component models and middlewares.

4.2 Adage

Adage [7] (*Automatic Deployment of Applications in a Grid Environment*) is a research project that aims at studying the deployment issues related to multi-middleware applications. One of its originality is to use a generic application description model (GADe) [10] to handle several middleware systems. Adage follows the deployment process described in this paper.

With respect to application submission, Adage requires an 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 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. However, the implemented schedulers, random and round-robin, do not take into account any control parameters but the constraints of the submission method. Processor architecture and operating system constraints are taking into account.

The generic application description model (GADe) provides a model close to the machines. It contains only four concepts: process, code-do-load, group of processes and interconnection [10]. Hence, this description format is independent of the nature of the application (i.e., distributed or parallel), but complete enough to be exploited by a deployment planning algorithm.

Adage supports multi-middleware applications through GADe and a plug-in mechanism. The plug-in is involved in the conversion from the specific to the generic application description but also during the execution phase so as to deal with specific middleware configuration actions. Translating a specific application description into the generic description turns out to be a straightforward task. Adage supports standard programming models like MPI (MPICH1-P4 and MPICH-G2), CCM and JXTA, as well as more advanced programming models like GridCCM.

Adage currently deploys only static applications. After the generic description is used by the planer to produce a deployment plan. Then, an enactment engine executes it and produces a deployment report which is used to produce two scripts: a script to get the status of deployed processes and a script to clean them up. There is not yet any dynamic support in Adage.

Adage supports resource constraints like operating system, processor architectures, etc. The resource description model of Adage takes into account (grid) networks with a functional view of the network topology. The simplicity of the model does not hinder the description of *complex* network topologies (asymmetric links, firewalls, non-IP networks, non-hierarchical topologies) [8]. A planer integrating such piece of information is being developed.

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 (MPI, (CCM, GridCCM, JXTA, etc.)
Multi-middleware application	Yes (in progress)	Yes
Dynamic application	Yes	No (in progress)
Resource constraints	Yes	Yes
Execution constraints	Yes	Yes
Grid Middleware	Many, via GAM (GT 2-4, and SSH)	SSH and GT2

4.3 Comparison of GEA and Adage

Table 1 sums up the similarities and difference between GEA and Adage with respect to the features of our common deployment process. The two prototypes are different approximations of the general model: GEA supports dynamic ASSIST applications. Dynamicity, instead, is not currently supported by Adage. On the other hand, multi-middleware applications are fully supported in Adage, as it is a fundamental requirement of GridCCM. Its support in GEA is in progress, following the incorporation of those middleware systems in the ASSIST component framework.

5. Conclusion

ASSIST and GridCCM programming models requires advanced deployment tools to handle both application and grid complexity. This paper has presented a common deployment process for components within a Grid infrastructure. This model is the result of several visits and meetings that were held during the last past months. It suits well the needs of the two projects, with respect to the support of heterogeneous hardware and middleware, and of dynamic reconfiguration. The current implementations of the two deployment systems – GEA and Adage– share a common subset of features represented in the deployment process. Each prototype implements some of the more advanced features. This motivates the prosecution of the collaboration.

Next steps in the collaboration will focus on the extension of each existing prototype by integrating the useful features present in the other: dynamicity in Adage and extending multi-middleware support in GEA. Another topic of collaboration is the definition of a common API for resource discovery, and a common schema for resource description.

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 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, Jan. 2006.

[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, pages 771–781, Lisboa, Portugal, August 2005. Springer.

[4] M. Danelutto, M. Vanneschi, C. Zoccolo, N. Tonellotto, R. Baraglia, T. Fagni, D. Laforenza, and A. Paccosi. HPC Application Execution on Grids. In V. Getov, D. Laforenza, and A. Reinefeld, editors, *Future Generation Grids*, CoreGrid series. Springer, 2006. Dagstuhl Seminar 04451 – November 2004.

[5] A. Denis, C. Pérez, and T. Priol. PadicoTM: An open integration framework for communication middleware and runtimes. *Future Generation Computer Systems*, 19(4):575–585, May 2003.

[6] F. Cappello, F. Desprez, M. Dayde, E. Jeannot, Y. Jegou, S. Lanteri, N. Melab, R. Namyst, P. Primet, O. Richard, E. Caron, J. Leduc, and G. Mornet. Grid'5000: A large scale, reconfigurable, controlable and monitorable grid platform. In *Grid2005 6th IEEE/ACM International Workshop on Grid Computing*, November 2005.

[7] S. Lacour, C. Pérez, and T. 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.

[8] S. Lacour, C. Pérez, and T. Priol. A Network Topology Description Model for Grid Application Deployment. In *the Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing (GRID 2004).* Springer, November 2004.

[9] S. Lacour, C. Pérez, and T. Priol. Description and packaging of MPI applications for automatic deployment on computational grids. Research Report RR-5582, INRIA, IRISA, Rennes, France, May 2005.

[10] S. Lacour, C. Pérez, and T. Priol. Generic application description model: Toward automatic deployment of applications on computational grids. In *the Proceedinfs of the 6th IEEE/ACM Int. Workshop on Grid Computing (Grid2005).* Springer, November 2005.

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

[12] C. Pérez, T. Priol, and A. Ribes. A parallel CORBA component model for numerical code coupling. *The Int. Journal of High Performance Computing Applications*, 17(4):417–429, 2003.

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

[14] D. Adami, M.Coppola, S. Giordano, D. Laforenza, M. Repeti, N. Tonellotto, Design and Implementation of a Grid Network-aware Resource Broker. In *Proc. of the Parallel and Distributed Computing and Networks Conf. (PDCN 2006).* Acta Press, February 2006.