Deploying on the Grid with DeployWare

Areski Flissi, Jérémy Dubus, Nicolas Dolet, Philippe Merle

Deploying on the Grid with DeployWare

Areski Flissi, Jérémy Dubus, Nicolas Dolet and Philippe Merle
GOAL/ADAM Team - **INRIA & LIFL, UMR 8022 USTL - CNRS**
{areski.flissi,jeremy.dubus}@lifl.fr,
{nicolas.dolet,philippe.merle}@inria.fr
59650 Villeneuve d'Ascq, FRANCE

Abstract

In this paper, we present DeployWare to address the deployment of distributed and heterogeneous software systems on large scale infrastructures such as grids. Deployment of software systems on grids raises many challenges like 1) the complexity to take into account orchestration of all the deployment tasks and management of software dependencies, 2) the heterogeneity of both physical infrastructures and software composing the system to deploy, 3) the validation to early detect errors before concrete deployments and 4) scalability to tackle thousands of nodes. To address these challenges, DeployWare provides a metamodel that abstracts concepts of the deployment, a virtual machine that executes deployment processes on grids from DeployWare descriptions, and a graphical console that allows to manage deployed systems, at runtime. To validate our approach, we have experimented DeployWare with a lot of software technologies, such as CORBA and SOA-based systems, on one thousand of nodes of Grid'5000, the french experimental grid infrastructure.

1 Introduction and Challenges

Grid computing [12] reality raises many challenges such as parallel computing, middleware, scheduling, peer to peer software for grids, security, administration of large scale, distributed and heterogeneous platforms and **deployment**. Deployment of distributed systems on grids is now a major research topic. This activity most of the time has to be done by system administrators, as users are not necessarily computer science researchers or engineers. But, deployment on grid infrastructures is a nightmare, even for system administrators. It can be defined as a set of tasks to orchestrate, such as connections to remote nodes, installation/uninstallation of middleware, software, useful libraries, and applications, configuration of nodes and software, starting/stopping of services or application servers, instantiation/killing of ap-

plications or programs, and data transfer. Particularly, in the context of *Grid Computing*, deployment raises many challenges:

Complexity - Due to the huge number of nodes that are potentially involved, deployment cannot be handled by humans. Elementary tasks that compose a deployment process have to be automatically orchestrated. For instance, the configuration of a software on a remote node obviously cannot be done before the software has been installed. Another example, which is specific to the Grid'5000 platform [7], is that a reservation submission request of nodes and/or clusters (using the OAR [6] batch scheduler) necessarily precedes any experiences using these nodes. Also, dependencies between software is a critical issue. Software most of the time depends on one or more software, which themselves can have complex dependencies. Currently, this problem is managed by hand by system administrators and often leads to errors or inconsistencies. Furthermore, deployment on large scale must allow the parallelization of some tasks (*e.g.*, the upload of a same software on a lot of nodes) whereas some other tasks have to be executed in a sequential order (*e.g.*, start a registry service before publishing any business services). At last, administration of a system that is composed of thousands of software distributed on the grid is hard for *human* administrators. This raises two problems: the monitoring of such a system and the management of it, once it has been deployed.

Heterogeneity - The second challenge is heterogeneity which exists at different levels. Firstly, it concerns the targeted physical infrastructure. Grids, clusters and in some cases the nodes that compose a cluster are heterogeneous in terms of hardware, network, operating systems or shells. Also, administrators/users have to deal with a lot of low-level deployment mechanisms such as remote access protocols (*e.g.*, SSH, Telnet, rlogin) and file transfer protocols (*e.g.*, FTP, HTTP, SCP). Secondly, heterogeneity concerns the software system to deploy itself. Software that compose the distributed system are heterogeneous in terms of technologies and/or paradigms. Distributed systems could

IEEE
computer
society

use different paradigms such as parallel programming with MPI, component-based software engineering (CBSE [24]), object-oriented programming (OOP) or service-oriented architectures (SOA [23]), but also a plethora of middleware for the grid exists, such as Globus [11], GridCCM [21], DIET [17] or ProActive [3], or runtime platforms such as SOA-based systems, JEE-based systems, OW2 Fractal-based systems [5], CORBA-based systems [18] (CORBA middleware, OpenCCM [19]). Furthermore, heterogeneity of software also concerns their granularity. Actually, software we consider here can be middleware, application servers, applications on top of middleware, objects or parallel components, services, libraries or binaries, and even virtual operating systems! The deployment procedures will strongly depend on both paradigm/technology and granularity.

Validation - Static validation is an important issue, especially for the deployment on thousands of nodes. Large scale deployment needs reliability, through a static validation before execution to prevent errors or inconsistencies. Before starting a deployment process, administrators must be sure of fulfilling all dependencies between software, and avoiding errors and resource conflicts. Then, how to deal with the problem of the sharing of file systems or port numbers by software/application servers?

Scalability - Scalability is the new challenge for *Grid Computing* community. Interconnections between grids, such as between the Naregi Japan grid [2] and the Grid'5000 french one, are becoming a reality and will increase the computational power for users. We have to deal with the limits of physical resources such as the number of open sockets on a single node, in order to achieve an automatic and parallel deployment on hundreds of thousands of nodes, and, in the future, even millions of nodes.

To address these four challenges, we propose DeployWare, a framework for the deployment of distributed and heterogeneous software systems on grids. DeployWare provides a *Domain Specific Modeling Language* (DSML) for the deployment based on a metamodel that captures abstract concepts of deployment and masks software heterogeneity from the user point of view. DeployWare models are validated before execution to ensure reliable deployments. DeployWare models, conform to our metamodel, describe configurations to deploy, regardless paradigms or technologies, and are mapped to concrete descriptions, written with an architecture description language (ADL). A distributed platform executes DeployWare descriptions thanks to a virtual machine that interprets these descriptions and automatically orchestrates the deployment process, and deal with software dependencies. The virtual machine is implemented using the CBSE approach. The deployment process of the system is reified as an assembly of software components. DeployWare can be distributed on multiple nodes for scalability
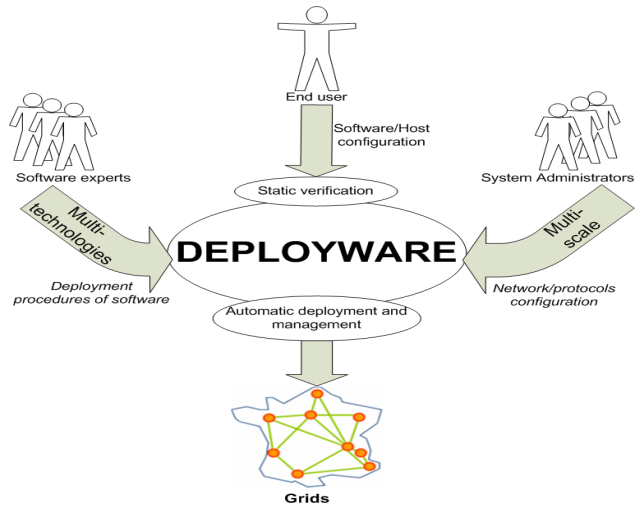

Figure 1. Overview of DeployWare

purpose. A graphical console allows one to monitor and manage the deployed system at runtime, by calling elementary deployment tasks on each software that composes it.

The remainder of this paper is organized as follows. Section 2 details our contribution to *Grid Computing*: the DeployWare framework and its elements. Experiments with DeployWare of large scale deployments on Grid'5000, the french experimental grid infrastructure, are shown in Section 3. In Section 4, related works are discussed and finally, Section 5 concludes this paper and gives some future works.

2 The DeployWare Framework

2.1 Overview

DeployWare is a framework for the description, the deployment and the management of heterogeneous and distributed software systems on grids. In order to address the challenges of deployment on grids, DeployWare provides 1) a DSML for the deployment based on a metamodel that captures abstract concepts of deployment, such as the notions of personality, software, procedure or instruction and a concrete syntax based on an ADL, 2) a virtual machine that interprets these concrete descriptions and executes the deployment process, plus a library of low-level deployment components masking the heterogeneity of the physical infrastructure, and 3) a graphical console that allows to manage/monitor, at runtime, systems that have been deployed with DeployWare.

Figure 1 gives an overview of the DeployWare framework. It shows the three actors involved in a DeployWare process: a *system administrator* describes the targeted infrastructure,

i.e. the nodes of the grid, the network and protocols, a *software expert* describes the deployment procedures of a given technology, the *end-users* simply declare the software to deploy and assign them to the nodes of the grid.

2.2 The DeployWare Metamodel

DeployWare lies on a metamodel (Figure 2). Its goal is to capture the abstract concepts of the deployment of a distributed system on a grid, independently of the underlying paradigm, technology or granularity. The DeployWare metamodel is composed of two main parts: the **TechnoExpert** package that defines the concepts that are manipulated by software experts of a given technology, and the **SystemAdmin** package, used by administrators/end-users to write models that represent their deployment configurations. A system can be seen as a set of software that have to be installed, configured, started and instantiated on a physical distributed infrastructure. So, the first key concept of the metamodel is **Personality**. It is used at design time to package a set of software related to a specific technology. A **Personality** is composed of **SoftwareType** elements that abstract the concept of *software* and represent any deployable artifact of a given technology. A software contains several generic deployment procedures such as install, configure, start, manage, stop, unconfigure and uninstall, symbolized by **Procedure** elements. Deployment procedures of any software mainly consist in executing some elementary actions that we call **Instruction**. For example, the *install* procedure of software most of the time consists of downloading/uploading an archive and extracting and installing it somewhere in the file system of the remote node. Software has generic configurable properties, as illustrated in Figure 2 with the **Property** element, such as a path to its archive or its home directory, but also specific ones, such as a port where a server must listen for client requests. Furthermore, software often have dependencies to other software. This is represented by the **dependencies** relation between **SoftwareType** elements. Software are declared, in a DeployWare model, at runtime and by end-users, using the **SoftwareInstance** concept and are assigned to physical nodes with the **HostInstance** element that represents a host.

In order to ensure reliable deployments, DeployWare models are validated before execution [9]. This validation is possible thanks to the DeployWare metamodel. Many conditions have to be fulfilled. First of all, all dependencies between software are checked. This means if a *software A* depends on a *software B*, DeployWare checks if an instance of *software B* is present for each instance of *software A* that has been declared by end-users. The second verification is about the **SoftwareType** procedures. For each proce-

Figure 2. The DeployWare Metamodel

dure, *e.g.* *install* or *start*, that has been declared by a software expert in a new **SoftwareType**, the *opposite* procedure must exist, *i.e.* respectively *uninstall* or *stop*. This ensures correct undeployment of the system, and to leave the nodes of the grid after the experiences as they were before. A verification is also made inside deployment procedures themselves, on the set of instructions. Procedures must be globally symetric. The goal of such a semantic validation is to prevent some unpredictable errors due to side effects. In support of this view, let us take a simple start procedure of a software, *e.g.* an application server, that contains an instruction that launches a daemon to start the server. If the stop procedure does not contain an instruction that kills the daemon, a sequence of start/stop deployment procedures on the software will inescapably lead to memory problem since the daemon is launched many time, but never killed. Finally, to deal with resource conflicts during the deployment process, some additionnal static verifications on ports and file systems are done. This allows to prevent for instance two servers from using the same port number on the same host.
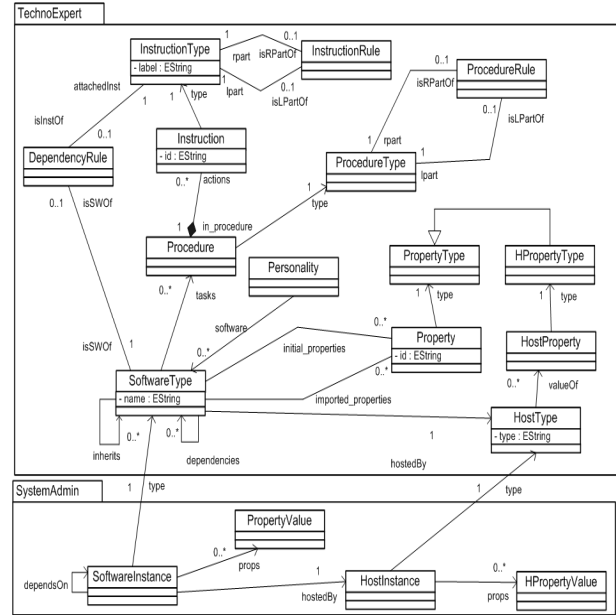
DeployWare models are mapped to DeployWare descriptions files that use a concrete syntax, based on an ADL allowing to describe a software system to deploy. These files hide complexity since end-users/administrators just have to describe the software system/grid infrastructure, instead of programming the deployment process. A DeployWare description is divided in two parts: the *hosts* part describes the infrastructure whereas the *software* part specifies the software to deploy and on which host, using identifiers that re-
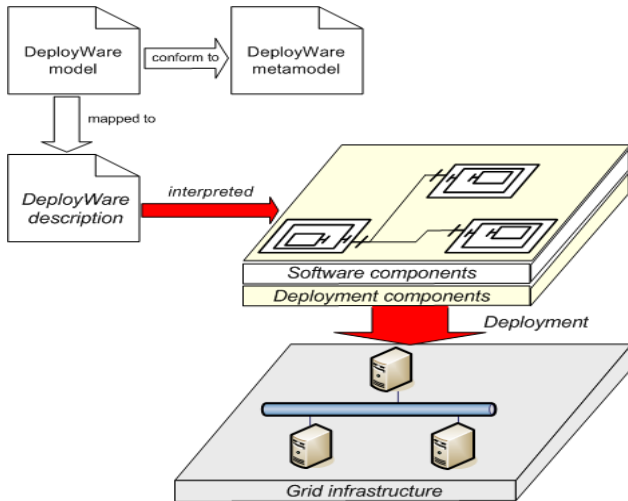
Figure 3. The DeployWare Virtual Machine

Figure 4. A DeployWare Software

fer to (*i.e.* are instance of) the `TechnoExpert` package's elements of the DeployWare metamodel. More information about the FDF syntax can be found in [10].

2.3 The DeployWare Runtime

The DeployWare runtime executes DeployWare descriptions. Its goal is to automatically execute complex deployment processes, dealing with orchestration and software dependencies, hardware heterogeneity and very large scale deployment.

2.3.1 A Virtual Machine to execute descriptions

The DeployWare runtime, named `Fractal Deployment Framework` (FDF), is a virtual machine that automatically executes and orchestrates the deployment process. Its implementation follows the CBSE approach, *i.e.* everything is reified as software components. FDF is composed of two layers: the *software components* layer and the *deployment components* one, as illustrated in Figure 3. The *software components* layer is the set of components that reify the high-level software to deploy. These components are the result of the mapping of software experts' DeployWare models, according to the Model-Driven Approach (MDA [22]). They define the personalities. The *deployment components* layer is a library of components that abstracts low-level deployment mechanisms, such as file transfer and remote access protocols, shells, etc. This allows us to mask the heterogeneity of the physical infrastructure. The FDF virtual machine interprets DeployWare descriptions as assemblies of components. These assemblies hide, from user's point of view, the complexity of the deployment process, and particularly
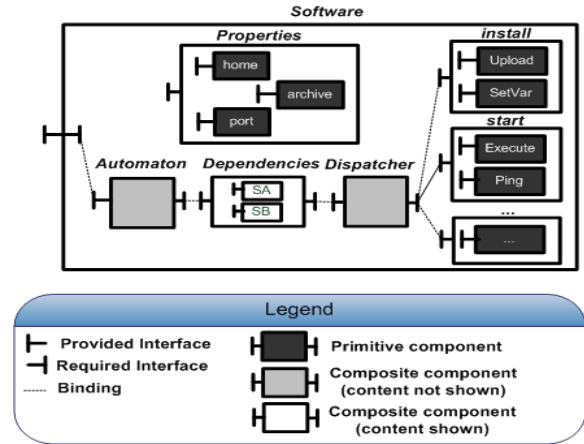
problems of the software dependencies and orchestration. Indeed, a software is represented as a composite of components as shown in Figure 4. Everything in FDF being components, `properties` is the composite that contains the configurable properties of a software, `dependencies` is the composite that contains references to other software components whereas the procedures, such as `install`, `configure` or `start`, are also composites of components symbolizing the instructions. The instructions are runnable components that use the *deployment components* library to realize elementary deployment tasks, such as uploading an archive, executing a remote shell command, setting an environment variable, etc.

Dependencies are automatically resolved thanks to an internal automaton. If a *software A* depends on a *software B*, *i.e.* a reference to *software B* is included in the `dependencies` composite of *software A*, then procedures of *software B* are executed before procedures of *software A*. In other words, starting *software A* means: starting *software B* then starting *software A*. It is also possible to associate different procedures in the case of, for example, starting the *software A* just requires the installation of *software B*. Obviously, dependencies of *software B* are also resolved recursively. Dependencies must be correctly declared in DeployWare models by software experts. Thus, the orchestration of the deployment of a system composed of various software that have complex dependencies are automatically managed by FDF, no more by users. The FDF implementation [10] is based on the Fractal component model [5] and its Java-based tools.

2.3.2 About Scalability

The FDF virtual machine, installed on one node of the grid, executes, from a DeployWare description, the deployment process on the other nodes of the grid. DeployWare pro-
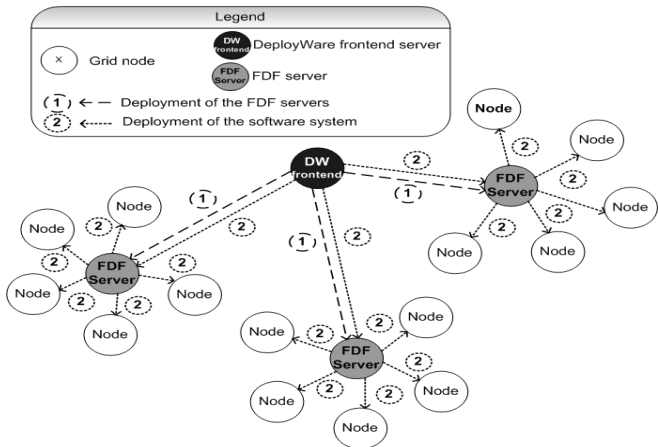
Figure 5. Distribution of DeployWare

Figure 6. The DeployWare eXplorer console

vides a way to specify the execution mode, parallel or sequential, of the deployment process for any set of software. It has been successfully tested for the deployment of the OpenCCM middleware on one thousand of nodes of Grid'5000 (see Section 3). But, to address very large scale deployments, *e.g.* on *5 000*, *10 000* or *50 000* nodes, the physical resources of the node that runs the FDF virtual machine are limited. These physical resources' limitations are, for example, the number of sockets that can be opened on a single node, the available memory, the CPU usage or the number of running threads. The idea then is to use multiple DeployWare nodes, by starting FDF on several nodes. For example, to deploy a software system on *10 000* grid nodes, *10* FDF *servers* can be used. Each FDF *server* is in charge of the deployment of a part of the system (on *1000* nodes), as illustrated in Figure 5. No bootstrap is required on the grid, the deployment of these FDF servers is accomplished using FDF itself! Users just specify the number of servers in DeployWare descriptions. The DeployWare frontend server, that deploys the FDF servers (step 1), launches and orchestrates the deployment of all the subsystems (step 2) by each FDF server.

2.4 The DeployWare eXplorer

The DeployWare eXplorer console addresses the administration issue. It is a graphical user interface allowing one to load DeployWare descriptions, as illustrated in Figure 6. Administrator can 1) explore hierarchically the system description to monitor it using the left tree panel, and 2), act on this system to manage it, using contextual menus. Figure 6 illustrates a deployment of OpenCCM application servers on Grid'5000. For a selected software, the console displays a graph (right panel) that shows all dependencies for the software and the node on which it is deployed. The ad-
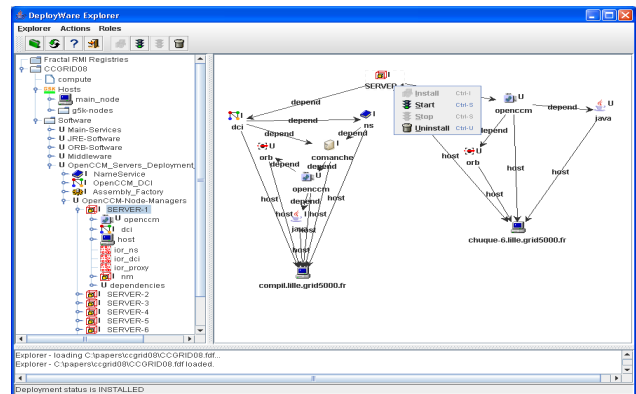
ministrator manages the system by calling the deployment procedures (such as install, configure, start, stop, uninstall) for any software declared in the DeployWare description, and can follow the progress of the whole deployment process thanks to icons that indicate the current state of the software (uninstalled, started, stopped, etc.).

3 Experiments on Grid'5000

3.1 Experiments with DeployWare

To validate our work, we have experimented it with a lot of paradigms, technologies, middleware, and applications. For that, we have written, as software experts, the personalities for many software such as:
- CORBA-based systems with CORBA middleware, the OW2 (previously, the *ObjectWeb* consortium) OpenCCM platform and CORBA component-based applications,
- Grid services with the OAR/OARGrid tool,
- SOA-based systems with SCA (*Service Component Architecture*) applications, Apache Tuscany SCA platform, BPEL (*Business Process Execution Language*) processes, OW2 Orchestra and ActiveBPEL engines, the OW2 PEtALS JBI (*Java Business Integration*) container and JBI components and assemblies,
- JEE-based systems with the Apache Geronimo, JBoss, JOnAS and SUN GlassFish application servers, and the JASMINe/Jade autonomic systems.
The complete list of supported software can be found at `http://fdf.gforge.inria.fr/`. Thus, writing a DeployWare personality (*e.g.*, to integrate a new software) is easy. There are two ways to achieve that: by designing DeployWare models that are mapped to DeployWare descriptions, as explained on Section 2.2, or simply using the FDF ADL syntax as illustrated in Figure 7. This piece of code shows the personality for LAM-MPI software. It details the *install* and *start* deployment procedures.

```
1   MPI.LAM = FDF.Software{
2     properties {
3         lamhosts-file = PARAM.HOME;
4         archive = PARAM.ARCHIVE;
5         home = PARAM.HOME;
6     }
7     install {
8         upload-archive = TRANSFER.Upload(#[archive],#[home]);
9         unarchive = SHELL.Unarchive(#[home]/#[archive]);
10        config-lam = SHELL.Execute(./configure --prefix=#[home]);
11        make = SHELL.Execute(make && make install);
12    }
13    configure { ... }
14    start {
15        start-rte = SHELL.Execute(#[home]/bin/lamboot -v #[lamhosts-file]);
16    }
17    ...
18  }
```

Figure 7. An example of FDF description

3.2 Results

The first experiment we have conducted concerns the deployment of the OpenCCM middleware on 1000 nodes of seven clusters of the Grid'5000 platform. The deployment of OpenCCM application servers means the deployment of the following software stack (*i.e.* the dependencies) on each node: the OpenCCM libraries, the JacORB middleware and a JRE[1]. The DeployWare description of this scenario includes a reservation task of 1000 nodes with the OAR tool. Figure 8 shows the execution time of the deployment pro-

Figure 8. Deployment on Grid'5000 results

cess according to the number of grid nodes, for this experiment. It may be of interest to point out that the execution time of the deployment grows linearly with the number of nodes, approximatively until 700-800 nodes. This result is interesting as it demonstrates the limits discussed in Section 2.3.2 about the physical resources. The deployment process is executed on one node of the grid by FDF. The deployment of the system on one thousand of nodes is executed according to a parallel mode. This means one thousand of sockets have to be opened at the same time on the node from which the deployment is launched, to execute the deployment instructions (*e.g.* the upload instructions with scp or shell commands to start processes on remote nodes with

---

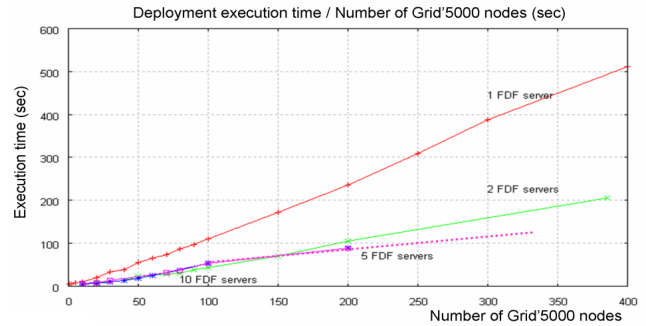[1]This represents approximatively 70 MB per node

Figure 9. Distributed deployment results

ssh). But, the number of opened sockets is fixed on the Grid'5000 nodes. Thus, sockets cannot be opened anymore if this limit is reached, until some sockets are closed, *i.e.* deployment instructions that use sockets are finished. That's the reason why the execution time starts to grow exponentially from 700-800 grid nodes, as illustrated in Figure 8 with the vertical bar. The second experiment is based on the same deployment process, but using multiple DeployWare nodes. This is so a *distributed* deployment. Figure 9 gives the results on 400 grid nodes for 1, 2, 5 and 10 FDF servers (unfortunately, we were not able to reserve more than 400 Grid'5000 nodes at this time). It shows that the deployment execution time decreases with the number of FDF servers, and this is an expected result. For example, the use of 2 FDF servers to deploy the system on 400 nodes of Grid'5000 has divided the execution time by 2. Nevertheless, we can notice that the use of more FDF servers, such as 10 servers to deploy on 100 nodes, is not really interesting as the number of grid nodes is not significant. We deduce that the use of one FDF server per 500 grid nodes can be a good compromise. We hope than experiments we plan to make in a near future, with at least 2000 nodes and 10 FDF servers, will reinforce this observation.

4 Related Work

ProActive [3] proposes a framework to program, compose and deploy distributed applications on grid infrastructures. **GoDIET** [8] is a deployment tool for distributed middleware. In those two approaches, efforts have been brought to wrap low-level network mechanisms and then to be able to automatically deploy both middleware and applications. Nevertheless, both ProActive and GoDIET are linked to their underlying technologies. ProActive allows to deploy ProActive-based servers and applications, whereas GoDIET is for the deployment of the DIET middleware. DeployWare is not linked to a specific technology or programming model and allows one to deploy any kind of software and middle-

ware. Besides, DeployWare can deploy the ProActive and DIET middleware and applications.

Jade [4] is a deployment approach that provides additional mechanisms to administrate systems autonomously according to *autonomic computing* principles. To deploy and administrate software with Jade, one has to wrap legacy code into Fractal components. Jade allows one to deploy any software in any language, and even black boxed legacy software. But, the wrapper of these legacy software requires Java and advanced wrapping techniques knowledge. The design of new personalities in DeployWare is done seamlessly using the very simple dedicated language. Moreover, Jade requires additionnal operations performed by hand to organise the deployment, such as starting Jade daemons/servers, registering them in a naming service, etc. DeployWare has no prerequisites for deploying any software. Besides, Jade belong to the list of supported DeployWare personalities.

GADe / ADAGE [15] proposes a generic process in the purpose of automatic deployment on *computational grids*. One of the main ideas is that *application* descriptions must be separated from *resource* descriptions. Both are inputs of the deployment planning phase which consists in mapping the application processes onto the selected computation nodes. Deployment plans are then executed by the ADAGE deployment tool. This generic process has been specialized for the deployment of CORBA component-based applications [13] and MPI applications [14] which are an important class of grid applications. DeployWare, through its metamodel, refines the three phases that are considered by the GADe / ADAGE architecture, and which are *i)* files uploading/installation, *ii)* processes launching and *iii)* application configuration, and validates some aspects of the deployment process. Nevertheless, we consider that the idea of the separation of application and resource descriptions could be really interesting for DeployWare, for example by defining a *virtual host* concept in our metamodel.

KaDeploy [1] is a tool allowing to execute deployment of operating system images on remote nodes. It is possible to build pre-configured images embedding every artifact that is necessary to a software system, and to automatically deploy these images onto each node of the grid infrastructure. For example, it can be used to deploy an image that include Globus on Grid'5000. Nevertheless, customizing and building OS images is a very costly activity. The configuration of the software and middleware contained in the image, and of the environment variables is really a complex task. Moreover, KaDeploy can't perform distributed orchestration of the deployment process. DeployWare is an alternate approach that allows to dynamically configure software and nodes, at runtime. **TakTuk** [16] or pssh are tools for deploying parallel remote executions of commands to a large set of remote nodes. TakTuk particularly addresses the scal-

ability challenge as the TakTuk engine can deploy itself on other nodes, like DeployWare. DeployWare can wrap such software to execute parallel low-level commands such as file transfers, environment settings or launching of remote processes. Nevertheless, pssh or TakTuk are mainly dedicated to parallel programs and are not able to orchestrate heterogeneous deployment tasks. Finally, the **OMG Deployment and Configuration of Component-based Applications (D&C)** [20] specification provides generic concepts to express deployment of component-based business applications, independently of the underlying componentmodel. Unfortunately, nothing is said about the deployment of the middleware/application server layer. Furthermore, the OMG D&C specification is not really adapted for SOA architectures. Additionally, the OMG D&C specification and its metamodel do not deal with verification, orchestration and scalability concerns, contrary to DeployWare.

5 Conclusion and Perspectives

In this paper, we have presented the DeployWare framework that particularly addresses the deployment of heterogeneous and distributed software systems on grids. DeployWare lies on a metamodel that captures abstract concepts of software deployment. The DeployWare metamodel is used to describe the deployment process of a system, independently of the underlying paradigm, technologies or middleware and runtime platforms that compose this system, and the granularity of the software, in order to deal with the software *heterogeneity* challenge. As grids are shared infrastructures, we are conviced that large scale deployments have to be validated before the execution. DeployWare provides this static validation. DeployWare models, that describe a set of software related to a specific technology (personality), are mapped to concrete software components, according the MDA principles. A concrete syntax allows to describe systems to deploy. The FDF virtual machine interprets these descriptions and the deployment process is automatically executed dealing with orchestration and software dependencies. FDF uses a library of deployment components that masks the heterogeneity of the physical infrastructures. The DeployWare eXplorer allows one to monitor and manage, at runtime, the deployed system. As far as *scalability* is concerned, DeployWare can deploy itself to achieve deployment on thousands of grid nodes. We validated our approach with concrete experiments of CORBA-based systems on the Grid'5000 platform. A lot of personalities, including component-based middleware, SOA-based systems, grid services or network tools are available yet. DeployWare/FDF is an LGPL open source project. Our future work will go in many directions. Firstly, we plan to make new experiments and measures of very large scale deployments using the whole Grid'5000 clusters, and if pos-

sible, the Naregi and DAS3 (the Netherlands grid initiative) grids. The purpose is to validate our approach, and particularly the distributed deployment using multiple Deploy-Ware nodes, and to experimentally verify the limits we identified. The scenario could be the automatic deployment of grid-specific software systems, such as MPI programs. Secondly, although DeployWare provides some mechanisms to dynamically compute hostnames for hosts, *e.g.* from a file that contains the list of nodes, software are statically assigned to hosts in end-users descriptions, contrary to grid's philosophy. The idea is to add an *assignation phase* to automatically select resources for software. Mid-term work concerns *autonomic computing*, faults tolerance and errors management (during the deployment process). We work on adding autonomic behaviour on deployment. For example, if new grid nodes become available, DeployWare will adapt the system, and automatically deploy software on these nodes. Finally, we think that it may be really interesting to simulate large scale deployment, before executing them on the grid. This means DeployWare models could be validated and executed on a virtual machine that symbolizes the grid infrastructure. Our long-term goal is to provide such a feature in DeployWare.

Acknowledgments

References

[1] *The KaDeploy project*. http://kadeploy.imag.fr/.

[2] *NAREGI: National Research Grid Initiative*, 2006. http://www.naregi.org.

[3] L. Baduel, F. Baude, D. Caromel, A. Contes, F. Huet, M. Morel, and R. Quilici. *Grid Computing: Software Environments and Tools*, chapter Programming, Deploying, Composing, for the Grid. Springer-Verlag, Jan. 2006.

[4] S. Bouchenak, N. D. Palma, D. Hagimont, and C. Taton. Autonomic Management of Clustered Applications. In *Cluster 2006*, Barcelona, Spain, Sept. 2006. IEEE.

[5] E. Bruneton, T. Coupaye, M. Leclercq, V. Quéma, and J.-B. Stefani. The FRACTAL Component Model and Its Support in Java. *Software Practice and Experience – Special issue on Experiences with Auto-adaptive and Reconfigurable Systems*, 36(11-12):1257–1284, Aug. 2006.

[6] N. Capit, G. D. Costa, Y. Georgiou, G. Huard, C. Martin, G. Mouni, P. Neyron, and O. Richard. A batch scheduler with high level components. In *CCGrid'05*, 2005.

[7] F. Cappello, E. Caron, M. Dayde, F. Desprez, E. Jeannot, Y. Jegou, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, P. Primet, and O. Richard. Grid'5000: a large scale, reconfigurable, controlable and monitorable Grid platform. In *Grid'2005 Workshop*, Seattle, USA, November 13-14 2005. IEEE/ACM.

[8] E. Caron, P. K. Chouhan, and H. Dail. Go DIET: A Deployment Tool for Distributed Middleware on Grid'5000. In *EXPGRID'06*, Paris, France, June 2006.

[9] J. Dubus and P. Merle. Towards Model-Driven Validation of Autonomic Software Systems in Open Distributed Environments. In *Proceedings of the ECOOP Workshop on Model-Driven Adaptation*, Berlin, Germany, July 2007.

[10] A. Flissi and P. Merle. A Generic Deployment Framework for Grid Computing and Distributed Applications. In *GADA'06*, volume 4279 of *LNCS*, pages 1402–1411, Montpellier, France, Nov. 2006. Springer-Verlag.

[11] I. Foster. Globus Toolkit Version 4: Software for Service Oriented Systems. In *IFIP NPC*, volume 3779 of *LNCS*, pages 2–13. Springer-Verlag, 2006.

[12] I. Foster and C. Kesselman. *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[13] S. Lacour, C. Pérez, and T. Priol. Deploying CORBA Components on a Computational Grid: General Principles and Early Experiments Using the Globus Toolkit. In *CD'04*, volume 3083 of *LNCS*, pages 35–49. Springer, 2004.

[14] S. Lacour, C. Pérez, and T. Priol. Description and Packaging of MPI Applications for Automatic Deployment on Computational Grids. Technical report, 2005. http://hal.inria.fr/inria-00070425/fr.

[15] S. Lacour, C. Pérez, and T. Priol. Generic Application Description Model: Toward Automatic Deployment of Applications on Computationnal Grids. In *Grid 2005, 6th Int'l Workshop*, Seattle, WA, USA, Nov. 2005. Springer-Verlag.

[16] C. Martin, O. Richard, and G. Huard. Déploiement adaptatif d'applications parallèles. *TSI*, 2005.

[17] J.-M. Nicod. DIET: Distributed Interactive Engineering Toolbox. In *Management of Metacomputers, Seminar N. 01241, Report N. 310*, Dagstuhl, Germany, May 2001.

[18] Object Management Group. CORBA Components Specification. Available Spec. formal/2006-04-01, Apr. 2006.

[19] Objectweb Consortium. *OpenCCM - The Open CORBA Component Model Platform*, 2002. http://openccm.objectweb.org.

[20] OMG. Deployment and Configuration of Component-based Distributed Applications. Available Spec. formal/2006-04-02, Apr. 2006.

[21] C. Pérez, T. Priol, and A. Ribes. A Parallel CORBA Component Model for Numerical Code Coupling. In *Grid 2002, 3rd International Workshop*, volume 2536 of *LNCS*, pages 88–99, Baltimore, MD, USA, Nov. 2002. Springer.

[22] J. D. Poole. Model-Driven Architecture: Vision, Standards And Emerging Technologies. In *ECOOP 2001, Workshop on Metamodeling and Adaptative Object Models*, Budapest, Hungary, Apr. 2001.

[23] M. Stal. Using Architectural Patterns and Blueprints for Service-Oriented Architecture. *IEEE Software*, 23(2):54–61, 2006.

[24] C. Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley, 2 edition, 2002.