Architecture-Based Autonomic Deployment of J2EE Systems in Grids

Didier Hoareau, Takoua Abdellatif, Yves Mahéo

Architecture-Based Autonomic Deployment of J2EE Systems in Grids

Didier Hoareau[1], Takoua Abdellatif [2], and Yves Mahéo[1]

[1] Valoria, University of South Brittany, France
{didier.hoareau|yves.maheo}@univ-ubs.fr
[2] ENISO, University of Sousse, Tunisia
takoua_abdellatif@yahoo.fr

Abstract. The deployment of J2EE systems in Grid environments remains a difficult task: the architecture of these applications are complex and the target environment is heterogeneous, open and dynamic. In this paper, we show how the component-based approach simplifies the design, the deployment and the reconfiguration of a J2EE system. We propose an extended architecture description language that allows specifying the deployment of enterprise systems in enterprise Grids, driven by resources and location constraints. With respect to these constraints we present a deployment process that instantiates propagatively the application, taking into account resources and hosts availability. Finally, we present an autonomic solution for recovery from failures.

1 Introduction

Grid environments have moved from the mere aggregation of computational resources dedicated to parallel and scientific applications to more general sharing of networked resources. The kind of Grids we consider in this paper can be seen as a set of heterogeneous machines interconnected by links of various capacities. Moreover a number of factors impacting the dynamism of the system (machine crashes, user disconnections, system failures etc.) cannot be neglected. Such Grids become attractive to multi-tier Internet service providers who want to improve the quality of service they offer. For this reason, many recent research works aim at finding the best models and techniques to exploit the Grids for better performance and high availability (e.g. [1, 2]). However, these works concentrate more on finding models and proving their effectiveness and do not propose efficient solutions automating the deployment and the recovery from failures of enterprise middleware and applications. Such features are very important and are still challenging in the context of interactive applications. Indeed, unlike scientific parallel applications whose parts can be independently deployed and executed, multi-tier middleware and applications are composed of interdependent pieces of software that have to coexist at execution time. Furthermore, the failure of one part of the enterprise system may involve service discontinuity or performance degradation. Recovering the system architecture, as initially defined at deployment time, is very important to preserve the agreed quality of service.

In this paper, we propose a solution for deploying enterprise systems in Grids and automating the recovery from failure of parts of the system. To achieve this goal, we

consider a J2EE system that we call a *virtual cluster*, similar to a classical J2EE cluster in that EJB and Web containers are replicated for backup fault-tolerance considerations. We believe that our solution is applicable to other models and other configurations of multi-tier Internet applications on wide-area networks, and it can be of interest to researchers in this field to easily experiment their different models on Grids and for service providers to easily handle an important number of clients. Our approach consists in applying an architecture-based deployment [3] and in automating the management of distributed systems. The idea is to abstract the managed system into an assembly of explicitly bound components and to use these components as units of configuration, deployment and reconfiguration. We adopted this approach for J2EE systems in a previous work—in classical cluster environments—by re-engineering an open source application server [4]. The re-engineering work consists in transforming the server parts into explicitly connected components. With the same component model, Fractal [5] in our case, we also represent the underlying resources like the nodes of the Grid. An ADL (Architecture Description Language) permits the description of the different parts of the distributed system, their configuration and their relations in terms of bindings and encapsulation. Finally, a deployment engine allows automating the deployment of the J2EE system using its description on the cluster targets. Compared to J2EE clusters, Grids are highly distributed, heterogeneous and dynamic. For this reason, our deployment system needs to be extended to manage virtual clusters within the Grid constraints. In this paper, we demonstrate the extension of the Fractal ADL to describe the component resources, a resource allocation mechanism and a solution for an automatic recovery from failures.

The layout of this paper is the following. In Section 2, we present more in details the context of our work and the main underlying assumptions. In Section 3, we describe our deployment process and its resource allocation service. We detail the current state of our implementation and some first results in Section 4. Section 5 discusses related work. Finally, Section 6 concludes the paper and identifies future work.

2 Context and main assumptions

2.1 J2EE system configuration and deployment

J2EE application servers are complex service-oriented architectures. In a previous work, we demonstrated that solving the deployment of J2EE applications requires that the internal software architecture of the J2EE server, in terms of the services that compose it and their various interaction and containment dependencies, be made explicit and modifiable at run time [4]. Indeed, the configuration of the system and its deployment parameters have to be described using the elements of the system's architecture. This description can then be used as a basis to implement and automate different deployment and reconfiguration policies. This is what is generally called *architecture-based management* [3]. For this purpose, we created JonasALaCarte, obtained by re-engineering the JOnAS (Java Open Application Server[3]) open source application server using the Fractal component model [5].

[3] http://jonas.objectweb.org

Thanks to a componentization of the server itself, where all the services are encapsulated into Fractal components, the architecture of the server is explicit. Both the hardware and the software entities are represented by components.

2.2 Deployment in a J2EE cluster

Building a J2EE cluster consists in replicating the Web and EJB tiers for load balancing and fault tolerance. A front-end load balancer (generally a HTTP server like Apache) dispatches the HTTP requests to the containers. A group communication system allows the consistency between stateful data hosted in the containers to be maintained. In order to deploy a clustered JonasALaCarte, the administrator has to produce an architecture descriptor (written with an ADL) together with a deployment descriptor. The first one defines the architecture of JonasALaCarte as a set of interconnected components and the second one exhibits the resource requirements of each component. The instantiation of this description allows the application server components to be configured and deployed on the target machines in an automated manner. Unlike in current JOnAS clusters, the unit of replication in JonasALaCarte is the service component and not the whole server. This selective replication is important since the EJB containers and the Web containers are generally execution bottlenecks and we need more replicas for these services than for other ones (Registry service, Transaction service, etc).

Figure 1 presents an example of an architecture for a J2EE clustered application server. Notice that we abstract the deployment and the configuration of an application server cluster into the uniform handling of Fractal components. Besides, a cluster configuration is just a particular configuration of the application server where components are distributed and replicated (represented in greyed boxes) on different JVMs. The same management tools are used to manage a stand-alone server in a single JVM and to manage a cluster of servers.

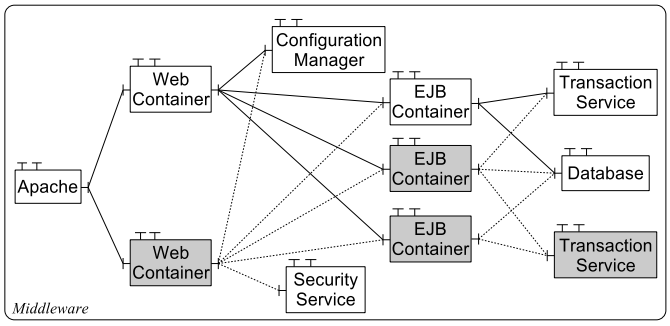

Fig. 1. Component-based view of JonasALaCarte in a cluster environment

2.3 From J2EE clusters management to *virtual clusters* management

We call a *virtual cluster* a J2EE system having the same configuration as a classical cluster (a front-end load balancer, a set of replicated containers and a group communication system for stateful data replication) but deployed in a Grid. By defining the number of replicas and the configuration of the services, the virtual cluster can represent different deployment models in wide-area networks. In this paper, we consider that our Grid system is composed of different zones; each zone groups a set of machines geographically close. Moreover, for each zone, some particular machines are well identified and are made public (on a Web site for example). We call *zone managers* these machines because they contribute in the deployment process.

Unlike a J2EE cluster, a Grid environment is highly distributed and are heterogeneous in terms of software and hardware configurations. Resource allocation is consequently a complex task. Grid machines are more dynamic either because they belong to end-users that frequently join and leave the Grid or because they are shared with other dynamic applications. However, if a machine involved in the execution of a multi-tier application leaves the system, a service discontinuity or a performance degradation may be induced leading to disastrous economic consequences. In front of these limitations, we identify the following requirements:

- Resource allocation should be automated. Each component has to explicitly define its required resources and the deployment system has to automatically find the appropriate target machine offering necessary resources for each component.
- Each variation in the Grid machines involved in an application execution has to be systematically detected and recovered. Indeed, in order to maintain the agreed quality of service, the configuration of the J2EE system has to be preserved. If the unavailable component is not replicated, its recovery allows ensuring the service continuity. In some cases, the service continuity is ensured thanks to the replication of the leaving component, like for containers. If the replica is a simple backup, this component needs to be replaced in order to preserve the fault-tolerance degree of the system and if the replica is involved in the load balancing, it also needs to be replaced to preserve the same level of performance.

3 Virtual cluster deployment system

In order to deploy a J2EE server system in a network such as the one described in Section 2.3, we cannot rely on a total knowledge of the different machines: this is hardly feasible as the size of a zone is important and as they are heterogeneous. Moreover, some machines—that were disconnected when the deployment was launched—can enter the network. Thus, traditional approaches, consisting in defining a target machine for each component of the application to be deployed, are not feasible in our context. We propose an extension to existing ADLs (xAcme[4], [6]) that allows the description of the resource properties that must be satisfied by a machine for hosting a specific component. In our approach, it is no more mandatory to give an explicit name or address

[4] http://www-2.cs.cmu.edu/ acme/pub/xAcme

of a target machine: the placement of components is mainly driven by constraints on the resources the target host(s) should satisfy. Then, we use the description of the architecture and the deployment specification to define a deployment of a J2EE system in a zone: installation and redeployment of the component are made in an automatic way.

In the following we present the general deployment algorithm in two steps. First, we describe the deployment process that allows the parts of the application to be deployed in a propagative way. Then, we present the mechanisms we have implemented to handle failures of the machines and of the different parts of the system.

3.1 Deployment specification

In order to specify the deployment of a J2EE system, we define two descriptor files written with FractalADL. The *architecture descriptor* contains the architecture of the system in terms of component definitions (their name, their client and server interfaces, their implementation) and component interactions (the bindings between components). The other descriptor, named *deployment descriptor*, contains, for each component, the description of the resources that the target platform must satisfy and references to component instances (defined in the architecture descriptor).

In the deployment descriptor a *deployment context* is defined for each component. Such a context lists all the constraints that a hosting machine has to verify. There are two types of constraints that can be defined in a deployment context: resource constraints and location constraints. Resource constraints allow hardware and software needs to be represented. Each of these constraints defines a domain value for a resource type that the target host(s) should satisfy. With location constraints some control on the placement of a component can be defined when more than one host applies for its hosting.

Figure 2 shows the deployment descriptor associated with the J2EE system represented in Figure 1 (Some repeated parts have been omitted). This descriptor contains the resource constraints associated with every component (e.g. lines 10–17: EJB container `ejb1` has to be installed on a host that have at least 512 MB of free memory) and location constraints, that indicate the co-location of some components (e.g. lines 45–47: transaction service component `transac1` must reside on the same host as the configuration manager since they share local resources in the current implementation). We can also control the location of a component according to the bandwidth of the network: lines 51–53 specify that the bandwidth between the machines hosting component `web1` and the others machines must be greater than 150 Mb/s).

For both performance scalability and high availability, each tier can be replicated. However, we should not require that all replicas be started at the same time. What is usually desired is to activate as soon as possible the Internet application when an EJB container is deployed and a Transaction Service is available. The other replicas, mainly used for performance, can be deployed later as soon as necessary resources become available. For this purpose, we have added a *cardinality* attribute to the description of a component's interface. This attribute takes the form of a couple of values that specify the minimum and the maximum number of bindings allowed through the interface.

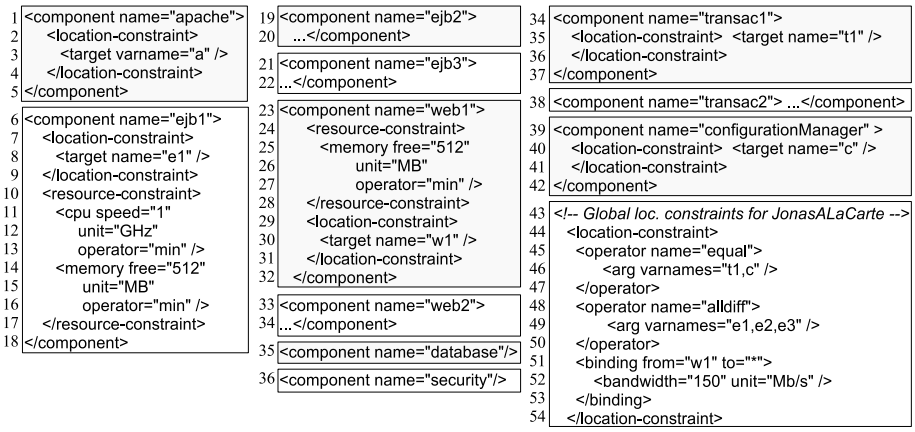

Fig. 2. Deployment descriptor of JonasALaCarte

3.2 Deployment process

As stated in section 2.3, dedicated machines—the zone managers—are defined for each zone. A given zone manager has two roles: (1) Maintaining a list of the machines in a zone and (2) orchestrating the deployment process in the zone.

We consider in this section a single manager per zone. The address of this manager is maintained on an already known site. A machine joining a zone gets the zone manager address and sends a presence notification message. The zone manager adds the newly connected machine in a list. The case of multiple zone managers, necessary for fault-tolerance, will be detailed in section 3.3.

The first step of the deployment process consists in sending the ADL files of the J2EE system to deploy to the zone manager (whose identity has been obtained beforehand by the administrator, from a given web site for example). As soon as the deployment descriptor is received by the manager, the deployment tasks are performed as follows:

1. The manager multicasts the deployment and architecture descriptors to all the zone nodes that are connected. The deployment descriptor contains resource and location constraints, and the identity of the manager.
2. Having received the deployment and architecture descriptors, each node checks the compatibility of its local resources with the resources required for each component. If it satisfies all the resource constraints associated with a component, it sends to the manager its candidature for the instantiation of this component.
3. The manager receives several candidatures and tries to compute a placement solution in function of the location constraints and the candidatures. In the case there is no location constraint associated with a component, the first candidate is chosen.
4. Once a solution has been found (or if a candidate has been chosen in the previous step), the manager updates the deployment descriptor with the new placement information and broadcasts it to all the zone nodes.

5. Each node that receives the new deployment descriptor updates its own one and is thus informed of which component it is authorized to instantiate and of the new location of the other components.
6. The final step consists in downloading necessary packages from well defined package repositories. The location of these repositories is defined in the deployment descriptor (not shown in the example for sake of clarity). For the components that are instantiated locally, their client interfaces (if any) must be bound to remote components. When the remote component possesses a constrained cardinality, a request is sent to the corresponding machine in order to know if a binding is possible. If the addition of a new binding is accepted at the server side and when a positive answer is received, the binding is achieved with the remote reference hold in the answer message. Besides, the number of incoming and outgoing binding is updated.

The above steps define a *propagative deployment*, that is, necessary components for running J2EE applications can be instantiated and started without waiting for the deployment of all the components in the ADL descriptor. As soon as a resource become available or a machine offering new resources will enter the network, candidatures for the installation of the "not yet installed" components will be sent to the zone manager, making the deployment progress.

When a new deployment descriptor is received (step 5) the binding establishment described at step 6 can also be made if the deployment descriptor contains new information on the location of some components that have to be bound with some already (locally) deployed components.

Let's consider an example of resource constraint. The constraint alldiff in the deployment descriptor (lines 48–49) indicates that the three EJBContainer must reside on three distinct hosts. In order to resolve this constraint, a machine must at least have the information of three machines that can hosts each one an EJBContainer. Thus, by collecting candidatures (step 3), the zone manager may decide on the placement of component provided there exists a combination of candidatures that solves the location constraints.

We can notice that in this deployment process: (1) the host selection of a component is made by the zone manager; (2) the instantiation of a component is achieved by the host selected by the zone manager; (3) the bindings needed by a component are initiated by the machine hosting it; (4) the activation of a component can be made as soon as its client interfaces are bound. Note that in our case, the activation of the container components (i.e. EJB and Web containers) involves the activation of the J2EE application running inside.

3.3 Automatic recovery from failures

In the environment we target, resources can also become unavailable (e.g. the amount of free memory demanded may decrease and become not sufficient), some parts of the J2EE system can be faulty, some machine may fail etc. In this paper, a failure can be due to a hardware crash of a machine, a disconnection from the network or a software bottleneck. This last case constitutes a failure of a component.

Failure of a component The recovery of a component and thus its redeployment consists in sending to the zone manager a message holding the identity of the component to redeploy. This is done by the machine hosting the faulty component (The failure, i.e. the non-responsiveness of the component, is detected through a probe associated with a control interface of the component.). Then, the zone manager updates the deployment descriptor by removing the location of the component and broadcasts the new descriptor to all the machines connected in the zone, automating the redeployment of the faulty component. Indeed, for all the machines, a component remains undeployed (i.e. it has no location), thus, they find themselves back in the propagative deployment. The phases of local evaluation of the resource constraints and the announcement of candidatures will go along.

When a component fails, it is important to consider its state. If the component is replicated, like the EJB container and the Web container services, the stateful data are automatically sent to any replica added to the group. This ensured by the group communication systems embedded within these components. Regarding the database, we consider that a regular copy is done on a data-center allowing to obtain stateful data when the database fails. This solution is frequently used in Internet applications deployed in wide-area networks, like in the edge-computing models.

When Apache fails, all the incoming requests are lost during the reconfiguration time. One solution consists in deploying a lightweight component storing the incoming requests in a list during the time the Apache component is recovering.

Resource violation When a resource constraint associated with a component is no longer verified on a specific host (for example the amount of free memory required is not sufficient), the corresponding component must be redeployed. This redeployment is performed the same way, except that the state of the component can be saved properly.

Failure of a machine other than a zone manager In a zone, a machine hosting one or several components may definitively crash. A crash is detected by the zone manager which maintains the list of the machine connected in the zone. When the manager detects a crash, as in the case of the failure of a component, it updates its deployment descriptor by removing the location of the component(s) that was running on the faulty machine. Then, the deployment descriptor is broadcast to other machines so that the missing components can eventually be re-instantiated.

Failure of a zone manager The crash of the zone manager is critical as it is responsible for choosing a host for each component. In order to deal with the failure of such a manager, we define several managers within a zone. Every manager has the same role as defined previously: it maintains the list of the machines that are connected in the zone; it collects the candidatures for the instantiation of components; and it resolves the location constraints depending on the received candidatures. To ensure the fault-tolerance of the zone manager, we consider a number of replicas. At a given time, a *leader* is in charge of establishing the deployment process. The address of the zone manager is mentioned in the deployment descriptor sent to the machines of the zone. Each information received by the leader is multicast to the backup managers using a group communication system offering the FIFO order and reliability. The failure of the leader is detected by

the backup machines and a new leader is elected. The zone manager identity is updated in the deployment descriptor and like any descriptor change, this piece of information is sent to the machines of the zone that will then deal with the new leader.

4 Implementation status and evaluation

4.1 Implementation status

The ADL presented in section 3.1 allows the specification of the placement of the components according to some conditions on resource and location constraints. We have chosen FractalADL to support the definition of deployment descriptors in an XML format. The main aspect with resource and location constraints are their manipulation at run time in order to observe and detect changes in the environment, to react on these changes and to find a placement solution at a given time according to some machine candidatures. We use Cream[5], a Java library for writing and solving constraint satisfaction problems or optimization problems, to represent interface cardinality, possible bindings and resource and location constraints.

Specific probes are used in order to introspect the resources needed by the components. We use DRAJE (Distributed Resource-Aware Java Environment) [7], an extensible Java-based middleware to model hardware resources (processor, memory, network interface...) or software resources (process, socket, thread...). For every resource constraint of the deployment descriptor, a resource in DRAJE is created and a periodic observation is launched. The value returned by a probe allows a host to check the consistency of a resource constraint according to the local resource state. If all the resource constraints associated with a component are verified by a machine, it applies for its instantiation. When the value returned by a probe does not respect a resource constraint, our run-time support is notified in order to redeploy the components that requires this resource as described in section 3.3. The current implementation of our system does not support the computation of bandwidths between machines but relies on a predefined file describing the properties of network links within a zone.

Component instantiation are made by a host when this host has been chosen by the zone manager. When an updated deployment descriptor is received, the location of the newly instantiated components is discovered, resulting in binding requests. When a binding is accepted, a stub component and a skeleton component are dynamically created thanks to the ASM library[6] and are deployed with FractalRMI. The server interfaces of the stub component are of the same type as the one of the local client interface that has to be bound. When the location of the EJBContainer is known, a new pair stub/skeleton is created and deployed if the number of outgoing bindings allowed (i.e. the interface cardinality) has not been reached.

4.2 Evaluation

A complete evaluation of the deployment and redeployment in the kind of environment we target implies to precisely control the dynamism of the different resources and hosts.

[5] http://kurt.scitec.kobe-u.ac.jp/~shuji/cream/
[6] http://asm.objectweb.org

We have indeed to take into account the announcement of machines' candidatures—which implies the availability of resources—in order to compute a placement solution. However the feasibility and the performance of the deployment process and recovery

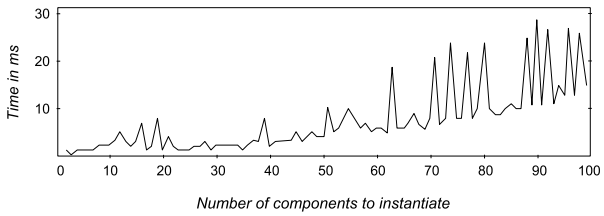

Fig. 3. Time required for a zone manager to decide on the placement of a set of components in function of the number of candidatures

mechanisms can be measured accurately when all the resources are available. In this case we can evaluate the time needed by a zone manager to compute a placement solution for the components of a virtual cluster.

Figure 3 shows the time for a zone manager to compute a placement solution when the number of received candidatures is sufficient, in function of the number of components to instantiate. We have considered a zone composed of a thousand of simulated machines on which the number of components to instantiate varies from one to one hundred. The experiment corresponds to the deployment of the architecture of Figure 1 according to the constraint "each component must reside on a distinct host" (alldiff constraint). Somewhat contrived, this constraint encompasses the complexity of other constraints involved in our deployment specification (resource constraints resolution has a negligible impact on the computation time). The evaluation has been conducted on a laptop (1,7 GHz Pentium Centrino). This experiment allowed us to verify that the time to compute—with the Cream library—a placement solution (when all conditions are met) remains acceptable regarding communication cost between machines. This computation time is likely not to be the prevalent factor in number of Grids configurations. We are currently conducting the evaluation of the deployment of a virtual cluster and the automatic management of failures on a Grid. The main difficult aspect remains the control of hosts and resources availability.

5 Related work

Our work is related to several different open-source and research domains. We single out the following ones: component-based deployment in Grid environments, multi-tier deployment in wide-area networks, resource allocation for distributed systems and architecture-based systems.

We share with GridCCM [8], GridKit [9] and Proactive [10] the same approach consisting in abstracting the system to deploy on the grids to an assembly of components. Proactive work is closer to ours since it considers Fractal component model to

represent hierarchical and parallel systems. However, our work covers both the resource management issues and the automatization of recovery from failures.

Exploiting the Grid resources to increase multi-tier application performance and fault-tolerance become recently the aim of many research teams [2, 1, 11]. However, focus is more on defining the best configuration and models to increase performance rather than on the management aspects.

Many works deal with resource allocation in distributed systems [12–15]. In our work, we propose a simple solution for resource allocation and we believe that, thanks to our modular component-model, we can easily adopt different policies and algorithms for an optimal resource usage. Furthermore, to our knowledge, most of the works on the Grids like PlanetLab and Globus, focus on parallel applications that are composed of independent tasks. Compared to the proposed solutions, we adopt an architecture-based approach motivated by the complex architecture of the multi-tier Internet application we address.

The architecture-based management approach [3] is mainly experimented in close environment like in SmartFrog [16] system or Jade system [17]. In these two systems, the deployment process considers that target machines are stable and homogeneous, which is not the case in Grids. Furthermore, handling failures relies on a centralized management unit, which hardly applies to the highly distributed Grid machines. In our solution, the machines collaborate in finding appropriate resources and for handling failures.

6 Conclusion

This paper proposes a solution for the deployment of enterprise systems in Grids and an automatic recovery management in face of failures. Deployment in such environment is quite challenging as the platforms we target are highly distributed, heterogeneous and dynamic. We offer a resource-aware deployment feature for J2EE systems, which is essential in Grid heterogeneous environments. We also demonstrate that the constraint-resolution is performed in a reasonable time. The role of the administrator is reduced to the writing of the deployment descriptor. All the deployment process and the recovery from failures are automated. Furthermore, the administrator does not need to be expert of the heterogeneous and complex J2EE systems. All the parts of the system are abstracted into Fractal components and the configuration is therefore unified. In our work, we aimed at maintaining the structure described in the ADL descriptor by replacing each time a faulty component by another. This allows ensuring the continuity of Internet services and maintaining their quality of service.

In this paper we adopted a special architecture of the J2EE system, the virtual clusters. We believe that our solution and mechanisms are applicable to other architectures. It is only necessary to write appropriate deployment descriptors and constraints. We are currently investigating a more complete evaluation of our approach on a Grid by taking into account resources and hosts availability. Moreover, some optimization can be defined when dealing with the placement decision of replicas by considering the symmetry of such components.

References

1. Rabinovich, M., Spatscheck, O.: Web Caching and Replication. Addison Wesley, Reading, Massachusetts, USA (2002)
2. Pierre, G., van Steen, M.: Globule: a Collaborative Content Delivery Network. IEEE Communications Magazine **44** (2006)
3. Dashofy, E., van der Hoek, A., Taylor, R.: Towards Architecture-Based Self-Healing Systems. In: Workshop on Self-Healing Systems, Charleston, South Carolina, USA (2002)
4. Abdellatif, T., Kornaś, J., Stefani, J.B.: J2EE Packaging, Deployment and Reconfiguration Using a General Component Model. In: Int. Working Conference on Component Deployment, Grenoble, France (2005)
5. Bruneton, E., Coupaye, T., Leclercq, M., Quéma, V., Stefani, J.B.: An Open Component Model and its Support in Java. In: Int. Symposium on Component-based Software Engineering, Edinburgh, Scotland (2004)
6. Dashofy, E., van der Hoek, A., Taylor, R.: An Infrastructure for the Rapid Development of xml-based Architecture Description Languages. In: Int. Conference on Software Engineering, Orlando, Florida, USA (2002)
7. Mahéo, Y., Guidec, F., Courtrai, L.: A Java Middleware Platform for Resource-Aware Distributed Applications. In: Int. Symposium on Parallel and Distributed Computing, Ljubljana, Slovenia (2003)
8. Denis, A., Pérez, C., Priol, T., Ribes, A.: Padico: A Component-Based Software Infrastructure for Grid Computing. In: Int. Parallel and Distributed Processing Symposium, Nice, France (2003)
9. Cai, W., Coulson, G., Grace, P., Blair, G.A., Mathy, L., Yeung, W.K.: The Gridkit Distributed Resource Management Framework. In: European Grid Conference, Amsterdam, The Netherlands (2005)
10. Baude, F., Caromel, D., Morel, M.: From Distributed Objects to Hierarchical Grid Components. In: Int. Symposium on Distributed Objects and Applications, Catania, Italy (2003)
11. Sivasubamanian, S., Alonso, G., Pierre, G., van Steen, M.: GlobeDB: Autonomic Data Replication for Web Applications. In: Int. World-Wide Web Conference, Chiba, Japan (2005)
12. Aron, M., Druschel, P., Zwaenepoel, W.: Cluster reserves: a mechanism for resource management in cluster-based network servers. In: Conference on Measurement and Modeling of Computer Systems, Santa Clara, California, USA (2000)
13. Appleby, K., Fakhouri, S., Fong, L., Goldszmidt, G., Kalantar, M., Krishnakumar, S., Pazel, D., Pershing, J., Rochwerger, B.: Oceano - SLA based management of a computing utility. In: Int. Symposium on Integrated Network Management, Seattle, Washington, USA (2001)
14. Fu, Y., Chase, J., Chun, B., Schwab, S., Vahdat, A.: SHARP: an architecture for secure resource peering. In: Symposium on Operating Systems Principles, Bolton Landing, New York, USA (2003)
15. Chase, J., Irwin, D., Grit, L., Moore, J., Sprenkle, S.: Dynamic Virtual Clusters in a Grid Site Manager. In: Int. Symposium on High Performance Distributed Computing, Seattle, Washington, USA (2003)
16. Goldsack, P., Guijarro, J., Lain, A., Mecheneau, G., Murray, P., Toft, P.: SmartFrog: Configuration and Automatic Ignition of Distributed Applications. In: Plenary Workshop of the HP OpenView University Association, Geneva, Switzerland (2003)
17. Bouchenak, S., Boyer, F., Hagimont, D., Krakowiak, S., Mos, A., de Palma, N., Quéma, V., Stefani, J.B.: Architecture-Based Autonomous Repair Management: An Application to J2EE Clusters. In: Symposium on Reliable Distributed Systems, Orlando, Florida, USA (2005)