A First Step towards Autonomous Clustered J2EE Applications Management

Slim Ben Atallah[1] Daniel Hagimont[2] Sébastien Jean[1] Noël de Palma[1]

[1] Assistant professor [2] Senior researcher

INRIA Rhône-Alpes – Sardes project

655 avenue de l'Europe , Montbonnot Saint Martin

38334 Saint Ismier Cedex, France

Tel : 33 4 76 61 52 00, Fax : 33 4 76 61 52 52

first.last@inrialpes.fr

ABSTRACT

A J2EE application server is composed of four tiers: a web front-end, a servlet engine, an EJB server and a database. Clusters allow for replication of each tier instance, thus providing an appropriate infrastructure for high availability and scalability. Clustered J2EE application servers are built from clusters of each tier and provide the J2EE applications with a transparent view of a single server. However, such applications are complex to administrate and often lack deployment and reconfiguration tools.

This paper presents JADE, a java-based environment for clustered J2EE applications deployment. JADE is the first attempt of providing a global environment that allows deploying J2EE applications on clusters. Beyond JADE, we aim to define an infrastructure that allows managing as autonomously as possible a wide range of clustered systems, at different levels (from operating system to applications).

General Terms

Management, Experimentation.

Keywords

Clustered J2EE Applications, Deployment, Configuration.

1. INTRODUCTION

J2EE-driven architectures are now a more and more convenient way to build efficient web-based ecommerce applications. Although this multi-tiers model, as is, suffers from a lack of scalability, it nevertheless benefits from clustering techniques that allow by means of replication and consistency mechanisms to increase application bandwidth and availability.

However, J2EE applications are not really easy and comfortable to manage. Their deployment process (installation and configuration) is as complex as tricky, no execution monitoring mechanism really exists and dynamic reconfiguration remains a goal to achieve. This lack of manageability makes it very difficult to take fully advantage of clustering capabilities, i.e. expanding/collapsing replicas sets as needed, and so on and so forth…

This paper presents the first results of an ongoing project that aims to provide system administrator with a management environment that is as automated as possible. Managing a system means being able to deploy, monitor and dynamically reconfigure such a system. Our first experiments target the deployment (i.e. installation/configuration) of a clustered J2EE application. The contribution in this field is JADE, a java-based application deployment environment that eases administrator's job. We show how JADE allows deploying a real benchmark application called RUBIS.

The outline of the rest of this paper is as follows. Section 2 recalls clustered J2EE applications architecture and life cycle and shows the limits of existing deployment and configuration tools. Then, Section 3 presents JADE, a contribution to ease such application management by providing automatic scripting based deployment and configuration tools. Section 4 resets this work in a wider project that consists of defining a component-based framework for autonomous systems management. Finally, Section 5 concludes and presents future work.

2. ADMINISTRATION OF J2EE CLUSTERS: STATE-OF-THE-ART AND CHALLENGES

This introductory section recalls clustered J2EE applications architecture and life cycle before showing the limits of associated management tools.

2.1 Clustered J2EE Applications and their Lifecycle

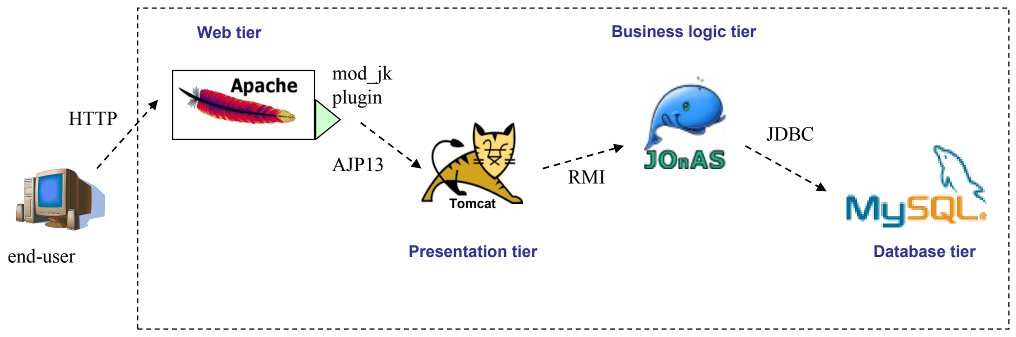

Figure 1. J2EE Applications Architecture

J2EE application servers [1], as depicted in Figure 1, are usually composed of four different tiers, either running on a single machine or on up to four ones:

- A **web tier**, as a web sever (e.g. Apache [2]), that manages incoming clients requests and, respectively depending if those relate to static or dynamic content, serves them or route them to the presentation tier using the appropriate protocol (e.g. AJP13 for Tomcat).

- A **presentation tier**, as a web container (e.g. Tomcat [3]), that receives forwarded request from the web tier, interacts with the business logic tier (using the RMI protocol) to get related data and finally dynamically generates a web document presenting the results to the end-user.

- A **business logic tier**, as an Enterprise JavaBeans server (e.g. JoNAS [4]), that embodies application logic components (providing them with non-functional properties) which mainly interact with the database storing application data by sending SQL requests by the way of the JDBC framework.

- A **database tier**, as a database management system (e.g.

MySQL server [5]), that manages application data.

The main motivations of clustering are scalability and fault-tolerance. Scalability is a key issue in case of web applications that must serve billion requests a day. Fault-tolerance does not necessarily apply to popular sites, even if it is also required in this case, but to applications where information delivery is critical (as commercial web sites for example). Both scalability and fault-tolerance are offered through replication (and consistency management for the last). In the case of J2EE applications, database replication provides application with service availability when machine failures occur, as well as efficiency by load balancing incoming requests between replicas.

The global architecture of clustered J2EE applications is depicted in Figure 2 and detailed below in the case of an {Apache, Tomcat, JoNAS, MySQL} cluster.

Apache clustering is managed through HTTP load balancing mechanisms that can involve hardware and/or software helpers. We cite below some well-known general-purpose techniques [6] that apply to any kind of web servers:

- Level-4 switching, where a high-cost dedicated router can distribute up to 700000 simultaneous TCP connections over the different servers
- RR-DNS (Round-Robin DNS), where a DNS server periodically changes the IP address associated to the web site hostname
- Microsoft's Network Load Balancing or Linux Virtual Server that use modified TCP/IP stacks allowing a set of hosts to share a same IP addresses and cooperatively serve requests
- TCP handoffs, where a front-end server establishes TCP connections and lets a chosen host directly handle the related communication.

Tomcat clustering is made by using the load balancing feature of Apache's *mod_jk* plugin. Each mod_jk can be configured in order to balance requests on whole or of a subset of Tomcat instances, according to a weighted round-robin policy.

No common mechanism exists to manage business logic tiers replicas, but ad'hoc techniques have been defined. For example, JoNAS clustering can be achieved by using a dedicated "cluster" stub instead of the standard RMI stub in Tomcat in order to interact with EJB. This stub can be seen as a collection stub that manages load balancing, assuming that whatever the JoNAS instance where a bean has been created, its reference is bound in all JNDI registries.

Database clustering solutions often remain commercial, like Oracle RAC (Real Application Cluster) or DB2 cluster and require using a set of homogeneous full replicas. We can however cite C-JDBC [7], an open source JDBC clustering middleware that allows using heterogeneous partial replicas providing with consistency, caching and load balancing.

J2EE applications life cycle consists in three main steps that are detailed below: deployment, monitoring and reconfiguration.

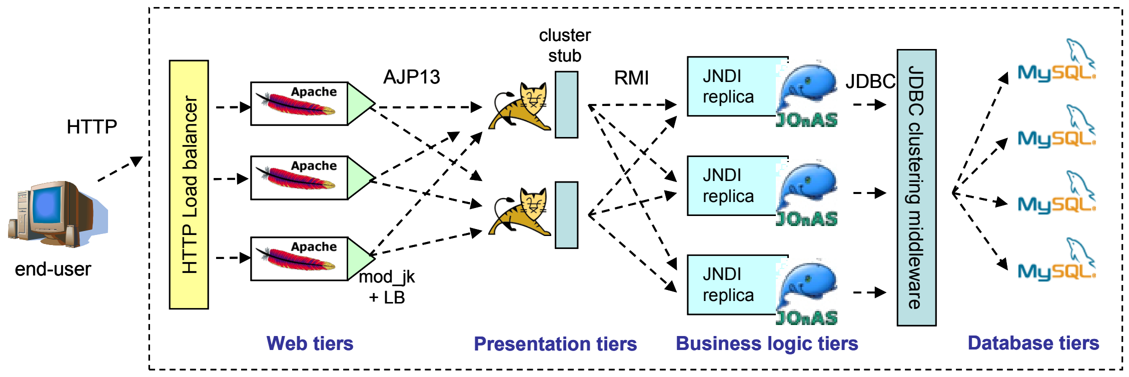

Figure 2. Clustered J2EE Applications Architecture

Deployment At the deployment step, tiers must firstly be installed on hosts and be configured to be correctly bound to each other. Then, application logic and data can be initialized. Application tiers are often delivered through installable packages (e.g. rpms) and the configuration is statically expressed in configuration files that statically map components to resources.

Monitoring Once the application has been deployed on the J2EE cluster, one needs to know both the system and the application states to be aware of problems that may arise. Most common issues are due either to hardware faults such as a node or network link failure, or inappropriate resource usage when a node or a tier of the application server becomes a bottleneck.

Reconfiguration Once a decision has been taken (e.g., extension of a J2EE tier on new nodes to handle increased load), one must be able to perform appropriate reconfiguration, avoiding as most as possible to stop the associated component.

2.2 Deployment, Monitoring and Reconfiguration Challenges

Currently, no integrated deployment environment exists for clustered J2EE applications. Each tier must be installed manually and independently. Identically, the whole assembly, including clustering middleware, must be configured manually mainly through static configuration files (and there also no configuration consistency verification mechanism). Consequently, the deployment and configuration process is a complex task to perform.
J2EE cluster monitoring is also a weakly offered feature. It is obviously possible to see hosts load or to use SNMP to track failures, but this is not enough to get pertinent information about application components.
There is no way to monitor an apache web server, and even if JoNAS offer JMX interfaces to see what applications are running, cluster administrator can not gather load evaluations at application level (but only the amount of memory used by the JVM). Finally, database servers usually do not offer monitoring features, except in few commercial products.
In terms of reconfiguration, no dynamic mechanism is really offered. Only Apache server enables to dynamically take into account configuration file changes, others tiers need to be stopped and restarted in order to apply low-level modifications.

In this context, in order to alleviate the burden of application administrator, to take advantage of clustering and thus to be able to optimize performance and resource consumption, there is a crucial need for a set of tools:

- an automated deployment and configuration tool, that allows to easily and user-friendly deploy and configure a entire J2EE application,
- an efficient application monitoring service that automatically gathers, filters, and notifies events that are pertinent to the administrator,
- a framework for dynamic reconfiguration.

Research work that is directly related to this topic is provided by the Software Dock [8.]. The Software Dock is a distributed, agent-based framework for supporting the entire software deployment life cycle.
One major aspect of the Software Dock research is the creation of a standard schema for the deployment process. The current prototype of the Software Dock system includes an evolving software description schema definition. Abstractly, the Software Dock provides infrastructure for housing software releases and their semantic descriptions at release sites and provides infrastructure to deploy or "dock" software releases at consumer sites. Mobile agents are provided to interpret the semantic descriptions provided by the release site in order to perform various software deployment life cycle processes. Initial implementations of generic agents for performing configurable content install, update, adapt, reconfigure, and remove have been created. However Software Dock does not deal with J2EE deployment as well as with clustered environment.

3. JADE: J2EE APPLICATIONS DEPLOYMENT ENVIRONMENT

In this section, we present JADE, a deployment environment for clustered J2EE applications. We firstly give an overview of the architecture and follow with the example of a benchmark application deployment called RUBIS.

3.1 Architecture Overview

JADE is a component-based infrastructure which allows the deployment of J2EE applications on cluster environment. As depicted in Figure 3, JADE is mainly composed of three levels defined as follows:

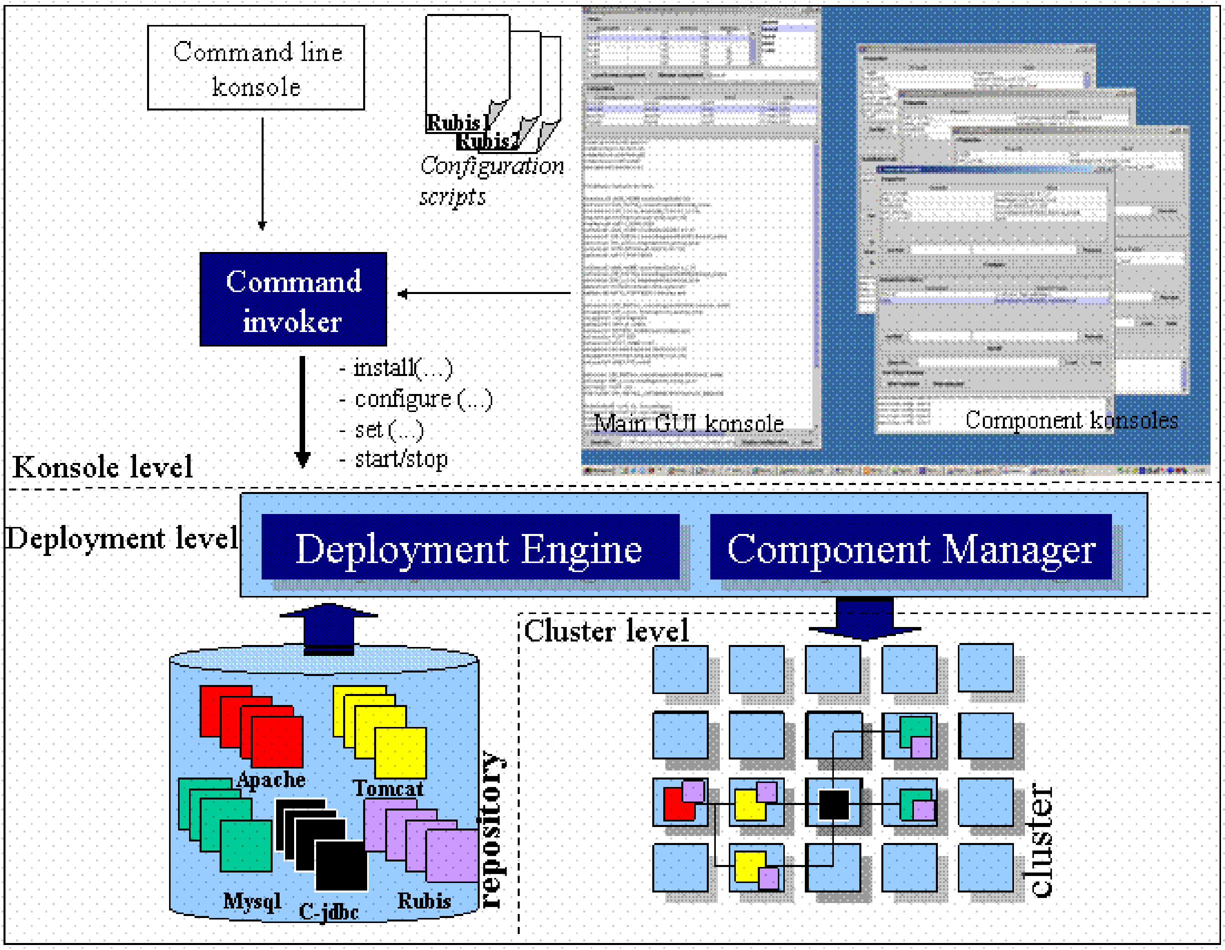

Figure 3. JADE Architecture Overview.

Konsole level

In order to deploy software components, JADE provides a configuration shell language. The language introduces a set of deployment commands described as follows:

- “start daemon”: starts a JADE daemon on a node
- “create”: creates a new component manager
- “set”: sets a component property
- “install”: installs component directories
- “installApp”: installs application code and data
- “start”: starts a component
- ”stop”: stops a component.

The use of configuration commands is illustrated in the RUBIS deployment use case in Appendix.

The Shell commands are interpreted by the *command invoker* that builds deployment requests and submit them to the *deployer engine*. JADE provides a GUI konsole which allows deploying of software components of cluster nodes. As shown in Figure 3, each started component is managed through its own GUI konsole. The GUI konsole also allows managing existing configuration shells.

Deployment level

It describes component repository, deployment engine and component manager:

- The repository provides access to several software releases (Apache, Tomcat, …) and associated component managers. It provides a set of interfaces for instantiating deployment engine and component manager.
- The deployment engine is the software responsible of performing specific tasks of the deployment process on the cluster nodes. The deployment process is driven by the deployment tools using interfaces provided by the deployment engine.
- Component manager allows setting component properties at launch time and also at run time.

Cluster level

The cluster level illustrates the components deployed and started on cluster nodes. At this stage, deployed components are able to be managed.

The JADE deployment engine is a component-based infrastructure. It provides the interface required to deploy the application on the required nodes. It is composed by component factory and by components deployer on each node involved. When a deployment shell runs a script, it begins with the installation of component factories on required nodes and then interacts with factories to create component deployer. The shell can then execute the script invoking component deployer. Component factory exposes an interface to remotely create and destroy component managers. Components deployers are wrappers that encapsulate legacy code and expose interface that allows installing tiers from the repository onto the local node, configuring the local installation, loading the application from the repository on tiers, configuring the application and starting/stooping tiers and the application.

The JADE command invoker submits deployment and configuration requests to the deployment engine. Even if currently the requests are implemented as synchronous RMI calls to the deployment engine interface, other connectors (such as MOM) should be easily plugged in the future.

A standard deployment script can perform the following actions: install the tiers, configure a tier instance, load the application on tiers, configure the application, start tiers. An example of deployment script is given in Appendix. A standard undeployment script should stop the application and tiers and should uninstall all the artefacts previously installed.

3.2 RUBiS deployment scenario

RUBiS [9] provides a real-world example of the needs for improved deployment activities support. This example is used to design a first basic deployment infrastructure. RUBiS is an auction site prototype modelled after eBay.com that is used to evaluate application design patterns and application servers performance and scalability.

RuBis offers different application logic implementations. It may take various forms, including scripting languages such as PHP that execute as a module in a Web server such as Apache, Microsoft Active Server Pages that are integrated with Microsoft's IIS server, Java servlets that execute in a separate Java virtual machine, and full application servers such as an Enterprise Java Beans (EJB) server [22]. This study focuses on the Java servlets implementation.

Since we take the use case of RuBis in a cluster environment, we depict a load balancing scenario. In Appendix is presented a configuration implying two Tomcat servers and two MySQL servers. In this configuration, the Apache server is deployed on a node called *sci40*, the tomcat servers are on nodes called *sci41* and *sci42*, and finally the two MySQL servers are on nodes called *sci43* and *sci44*.

4. TOWARDS A COMPONENT-BASED INFRASTRUCTURE FOR AUTONOMOUS SYSTEMS MANAGEMENT

The environment presented in the previous section is suitable for J2EE application deployed but, more generally, it can be easily derived to be applied to system management.

4.1 Overview of System Management

Managing a computer system can be understood in terms of the construction of system control loops as stated in control theory [10]. These loops are responsible for the regulation and optimization of the behavior of the managed system. They are typically closed loops, in that the behavior of the managed system is influenced both by operational inputs (provided by clients of the system), and by control inputs (inputs provided by the management system in reaction to observations of the system behavior). Figure 4 depicts a general view of control loops that can be divided into multi-tier structures including: sensors, actuators, notification transport, analysis, decision, and command transport subsystems.

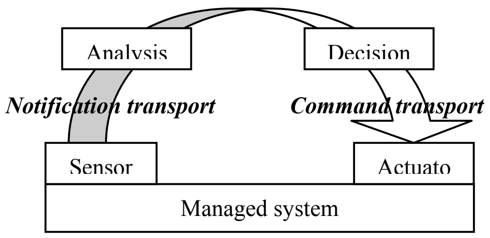

Figure 4. Overview of a Supervision Loop.

Sensors locally observe relevant state changes and event occurrences. These observations are then gathered and transported by ***notification transport subsystems*** to appropriate observers, i.e., analyzers. The ***analysis*** assesses and diagnoses the current state of the system. The diagnosis information is then exploited by the ***decision*** subsystem, and appropriate command plans are built, if necessary, to bring the managed system behavior within the required regime. Finally, ***command transport*** subsystems orchestrate the execution of commands required by the command plan while ***actuators*** are the implementation of local commands.

Therefore, building an infrastructure for system management can be understood as providing support for implementing the lowest tiers of a system control loop, namely the sensors/actuators and notification/command transport subsystems. We consider that such an infrastructure should not be sensitive as to how the loop is closed at the top (analysis and decision tiers), be it by a human being or by a machine (in the case of autonomous systems).

In practice, a control loop structure can merge different tiers, or have trivial implementations for some of them (e.g., a reflex arc to respond in a predefined way to the occurrence of an event). Also, in complex distributed systems, multiple control loops are bound to coexist. For instance, we need to consider horizontal coupling whereby different control loops at the same level in a system hierarchy cooperate to achieve correlate regulation and optimization of overall system behavior by controlling separate but interacting subsystems [11]. We also need to consider vertical coupling whereby several loops participate, at different time granularities and system levels, to the control of a system (e.g., multi-level scheduling).

4.2 Beyond JADE

JADE is a first tool that has to be completed by other ones that provide administration process with monitoring and reconfiguration.

Before this, a prerequisite is a cartography service that builds a comprehensive system model, encompassing all hardware and software resources available in the system. Instead of relying on a "manual" selection of resource eligible for hosting tiers, the deployment process should dynamically map application components on available resources by querying the cartography service that maintains a coherent view of the system state. A component-based model is well suited to represent a system model. Each resource (node, software …) can be represented by a component. Composition manifests hierarchical and containment dependencies.

With such an infrastructure, a deployment description no more needs to bind static resources but only needs to define the set of required resources. The architecture description might include an exact set of resources or just define a minimal set of constraints to satisfy. The cartography service can then inspect the system representation to find the needed components that correspond to the resources needed by the application. The deployment process itself consists in inserting the application components into the node component that contains the required resources. Finally, the application components are bound to the resources via bindings to reflect the resource usage in the cartography. The effective deployment of a component is then performed consecutively, as the component model allows some processing to be associated with the insertion or removal of a sub-component.

Then, there is also a need for a monitoring service reporting the current cluster state to take the appropriate actions. Such a monitoring infrastructure requires sensors and a notification transport subsystem.

Sensors can be implemented as components and component controllers that are dynamically deployed to reify the state of a particular resource (hardware or software). Some sensors can be generic to interact with resources through common protocols such as SNMP or JMX/RMI, but other probes are specific to a resource (processor sensor). Deploying sensors optimally for a given set of observations is an issue. Sensors monitoring physical resources may have to be deployed where the resource is located (e.g., to monitor resource usage) or on remote nodes (e.g., for detecting node failures). Another direct concern about sensors is their intrusiveness on the system. For instance, the frequency of probing must not significantly alter the system behavior. In the case of a J2EE cluster, we have to deal with different legacy software for each tier. Some software, such as web or database servers, do not provide monitoring interfaces, in which case we have to rely on wrapping and indirect observations using operating system or physical resource sensors. However, J2EE containers usually provide JMX interfaces that offer a way to instrument the application server. Additionally, the application programmer can provide user level sensors (e.g., in the form of JMX MBeans).

Notification transport is in charge of event and reaction dispatching. Once the appropriate sensors are deployed, they generate notifications to report the state of the resource they monitor. The notifications must be collected and transported to the observers and analyzers that have expressed interest in them. An observer can for instance be a monitoring console that will display the state of the system in a human readable form. Different observers and analyzers may require different properties from the channel used to transport the notifications. An observer in charge of detecting a node failure may require a reliable channel providing a given QoS, while these properties are not required by a simple observer of the CPU load of a node. Therefore the channels used to transport the notifications should be configured according to the requirements of the concerned observers and analyzers. Typically, it should be possible to dynamically add, remove, or configure a channel between sensors and observers/analyzers.

To this effect, we have implemented DREAM (Dynamic REflective Asynchronous Middleware) [12], a Fractal-based framework to build configurable and adaptable communication subsystems, and in particular asynchronous ones. DREAM components can be changed at runtime to accommodate new needs such as reconfiguring communication paths, adding reliability or ordering, inserting new filters and so on. We are currently integrating various mechanisms and protocols in the DREAM framework to implement scalable and adaptable notification channels, drawing from recent results on publish-subscribe routing and epidemic protocols.

5. CONCLUSION AND FUTURE WORK

As the popularity of dynamic-content Web sites increases rapidly, there is a need for maintainable, reliable and above all scalable platforms to host these sites. Clustered J2EE servers is a common solution used to provided reliability and performances. J2EE clusters may consist of several thousands of nodes, they are large and complex distributed system and they are challenging to administer and to deploy. Hence is a crucial need for tools that ease the administration and the deployment of these distributed systems. Our ultimate goal is to provide a reactive management system.

We propose the JADE tool which is a framework to ease J2EE applications deployment. Jade provides automatic scripting-based deployment and configuration tools in clustered J2EE applications. We experienced a simple configuration scenario based on a servlet version of an auction site (RuBiS). This experiment provides us the necessary feedback and a basic component to develop a reactive management system. It shows the feasibility of the approach. JADE is a first tool that provides with deployment facility, but it has to be completed to provide a full administration process with monitoring and reconfiguration.

We are currently working on several open issues for the implementation of our architecture system model and instrumentation for resource deployment, scalability and coordination in the presence of failures in the transport subsystem, automating the analysis and decision processes for our J2EE use cases. We plan to experiment JADE with other J2EE scenarii including EJB (The EJB version of RuBis). Our deployment service is a basic block for administration system. It will be integrated in the future system management service.

6. References

[1] S. Allamaraju et al. – Professional Java Server Programming J2EE Edition - *Wrox Press, ISBN 1-861004-65-6*, 2000.

[2] http://www.apache.org

[3] http://jakarta.apache.org/tomcat/index.html

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

[5] http://www.mysql.com/

[6] http://www.onjava.com/pub/a/onjava/2001/09/26/load.html

[7] Emmanuel Cecchet and Julie Marguerite. C-JDBC: Scalability and High Availability of the Database Tier in J2EE environments. In the 4th ACM/IFIP/USENIX International Middleware Conference (Middleware), Poster session, Rio de Janeiro, Brazil, June 2003.

[8] R.S. Hall et Al. An architecture for Post-Development Configuration Management in a Wide-Area Network. *In the 1997 International Conference on Distributed Computing Systems.*

[9] Emmanuel Cecchet, Anupam Chanda, Sameh Elnikety, Julie Marguerite and Willy Zwaenepoel. Performance Comparison of Middleware Architectures for Generating Dynamic Web Content. *In Proceedings of the 4th ACM/USENIX International Middleware Conference (Middleware), Rio de Janeiro, Brazil, June 16-20, 2003*

[10] K. Ogata – Modern Control Engineering, 3rd ed. – Prentice-Hall, 1997.

[11] Y. Fu et al. – SHARP: An architecture for secure resource peering – *Proceedings of SOSP'03.*

[12] Vivien Quéma, Roland Balter, Luc Bellissard, David Féliot, André Freyssinet and Serge Lacourte. Asynchronous, Hierarchical and Scalable Deployment of Component-Based Applications. *In Proceedings of the 2nd International Working Conference on Component Deployment (CD'2004), Edinburgh, Scotland, may 2004.*

7. APPENDIX

```
// start the daemon (ie : the factory)
start daemon sci40
start daemon sci41
start daemon sci44
start daemon sci45

// create the managed component:
// type name host
create apache apache1 sci40
create tomcat tomcat1 sci41
create tomcat tomcat2 sci42
create mysql mysql1 sci43
create mysql mysql2 sci44

// Configure the apache part
set apache1 DIR_INSTALL
/users/hagimont/apache_install
set apache1 DIR_LOCAL
/tmp/hagimont_apache_local
set apache1 USER hagimont
set apache1 GROUP sardes
set apache1 SERVER_ADMIN hagimont@imag.fr
set apache1 PORT 8081
set apache1 HOST_NAME sci40

//bind to tomcat1
set apache1 WORKER tomcat1 8009 sci41 100
// bind to tomcat2
set apache1 WORKER tomcat2 8009 sci42 100

set apache1 JKMOUNT servlet

// Configure the two tomcat
set tomcat1 JAVA_HOME
/cluster/java/j2sdk1.4.2_01
set tomcat1 DIR_INSTALL
/users/hagimont/tomcat_install
set tomcat1 DIR_LOCAL
/tmp/hagimont_tomcat_local

// provides worker port
set tomcat1 WORKER tomcat1 8009 sci41 100

set tomcat1 AJP13_PORT 8009
set tomcat2 DataSource mysql2
set tomcat2 JAVA_HOME
/cluster/java/j2sdk1.4.2_01

set tomcat2 DIR_INSTALL
/users/hagimont/tomcat_install

set tomcat2 DIR_LOCAL
/tmp/hagimont_tomcat_local


// provides worker port

set tomcat2 WORKER tomcat2 8009 sci42 100


set tomcat2 AJP13_PORT 8009

set tomcat2 DataSource mysql2


// Configure the two mysql

set mysql1 DIR_INSTALL
/users/hagimont/mysql_install

set mysql1 DIR_LOCAL
/tmp/hagimont_mysql_local

set mysql1 USER root

set mysql1 DIR_INSTALL_DATABASE
/tmp/hagimont_database


set mysql2 DIR_INSTALL
/users/hagimont/mysql_install

set mysql2 DIR_LOCAL
/tmp/hagimont_mysql_local

set mysql2 USER root

set mysql2 DIR_INSTALL_DATABASE
/tmp/hagimont_database
```

```
// Install the component

install tomcat1 {conf, doc, logs,webapps}

install tomcat2 {conf, doc, logs,webapps}

install apache1 {icons,bin,htdocs,cgi-
bin,conf, logs}

install mysql1 {}

install mysql2 {}


// Load the application part in the
middleware

installApp mysql1 /tmp/hagimont_mysql_local
""

installApp mysql2 /tmp/hagimont_mysql_local
""

installApp tomcat1
/users/hagimont/appli/tomcat rubis

installApp tomcat2
/users/hagimont/appli/tomcat rubis

installApp apache1
/users/hagimont/appli/apache Servlet_HTML


// Start all the component

start mysql1

start mysql2

start tomcat1

start tomcat2

start apache1
```