Component-Based Autonomic Management System

Jeremy Philippe[α], Sylvain Sicard[β] ,Christophe Taton[α]
[α]*INP Grenoble*
[β]*Université Joseph Fourier, Grenoble*
E-Mail : First.Last@inrialpes.fr

1. Introduction

Today's computing environments are becoming increasingly sophisticated. They involve numerous complex software that cooperate in potentially large scale distributed environments. These software are developed with very heterogeneous programming models and their configuration facilities are generally proprietary. Therefore, the administration of these software (installation, configuration, tuning, repair …) is a much complex task which consumes a lot of resources:

- human resources as administrators have to react to events (such as failures) and have to reconfigure (repair) complex applications,
- hardware resources which are often reserved (and overbooked) to anticipate load peaks or failures.

A very promising approach to the above issue is to implement administration as an autonomic software. Such a software can be used to deploy and configure applications in a distributed environment. It can also monitor the environment and react to events such as failures or overloads and reconfigure applications accordingly and autonomously. The main advantages of this approach are:

- Providing a high-level support for deploying and configuring applications reduces errors and administrator's efforts.
- Autonomic administration allows the required reconfigurations to be performed without human intervention, thus saving administrator's time.
- Autonomic administration is a means to save hardware resources as resources can be allocated only when required (dynamically upon failure or load peak) instead of pre-allocated.

This paper presents Jade, an environment for developing autonomic administration software. One main feature of Jade is that it relies on a component model to administer the environment as a component-based software. The component model provides the necessary architectural information to configure and reconfigure the environment in an autonomic manner, through control loops which link probes to reconfiguration services and implement autonomic behaviors. The administration software is also itself based on components to get the same autonomic capabilities.

We implemented a first prototype of the Jade environment and used it for deployment and repair management of a clustered J2EE application. In section 2, we present an overview of Jade. We conclude the paper in section 3 with a summary of our works in progress.

2. An overview of the Jade self-management system

2.1. Design of jade

We have adopted the overall organization proposed for autonomic computing [2]. The administration software is composed of components called Autonomic Manager, which implement a feedback control loop, regulating a part of a system, called a *Managed Resource*. In order to allow hierarchical control, an *Autonomic Manager* may itself play the role of a Managed Resource.

In order to be controllable, a Managed Resource is itself a component equipped with a management interface, which provides entry points for an Autonomic Manager. This interface should allow the manager to observe and to change the state of the resource.

The management interface needs to provide the following functions.

- Inspecting the contents of the managed resource, i.e. consulting any readable parameters; reading the values of any probes attached to resource; if the managed resource is composite (made of an assembly of parts), retrieving information on the structure of this assembly.
- Deploying and (re)configuring the managed resource; if again the resource is composite, modifying the structure of the assembly, e.g. dynamically inserting, rebinding, or removing some of its parts (for instance inserting a probe, adding a node to a cluster, etc.).

The component model that we use is Fractal [1], which has the following benefits.

- It provides a uniform, adaptable, control interface that allows introspection (observing the properties of the component) and dynamic binding (reconfiguring an assembly of components).
- It defines a hierarchical composition model for components, allowing a sub-component to be shared between enclosing components, at any level of granularity.
- Wrapping legacy entities within components allows to provide a uniform configuration interface, instead of relying on resource-specific, hand-managed, configuration files.
- Managing complex environments with different points of view. For instance, using appropriate wrapping components, it is possible to represent the network topology, the configuration of the J2EE middleware, or the configuration of an application on the J2EE middleware.
- Adding a control behavior to the encapsulated legacy entities (e.g. monitoring, interception and reconfiguration).

2.2. Self-management for J2EE applications

The above approach is illustrated in the case of a J2EE architecture. In this setting, an L5-switch balances the requests between two Apache server replicas. The Apache servers are connected to two Tomcat server replicas. The Tomcat servers are both connected to the same MySQL server.

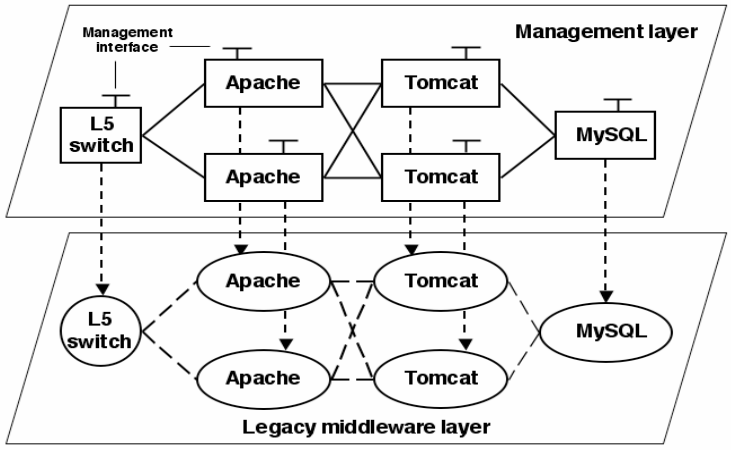

Figure 1 – Component-based management

In the above figure, the vertical dashed arrows represent management relationships between components and the wrapped software entities. In the legacy layer, the dashed lines represent relationships (or bindings) between legacy entities, whose implementations are proprietary. These bindings are represented in the management layer by component bindings (full lines in the figure).

Wrapping managed resources

In the management layer, all components provide the same (uniform) management interface for the encapsulated resources, and the corresponding implementation is specific to each resource (e.g., in the case of J2EE, Apache, Tomcat, MySQL, etc.). The interface allows managing the resource's attributes, bindings and lifecycle.

Relying on this management layer, sophisticated administration programs can be implemented, without having to deal with complex, proprietary configuration interfaces, which are hidden in the wrappers. The management layer provides all the facilities required to implement such administration programs:

- *Introspection:* The framework provides an introspection interface that allows observing managed resources (MR). For instance, an administration program can inspect an Apache MR (encapsulating the Apache server) to discover that this server runs on *node1:port 80* and is bound to a Tomcat server running on *node2:port 66*. It can also inspect the overall J2EE infrastructure, considered as a single MR, to discover that it is composed of two Apache servers interconnected with two Tomcat servers connected to the same MySQL server.
- *Reconfiguration*: The framework provides a reconfiguration interface that allows control over the component architecture. In particular, this control interface allows changing component attributes or bindings between components. These configuration changes are reflected onto the legacy layer. For instance, an administration program can add or remove an Apache replica in the J2EE infrastructure to adapt to the current load.

The implementation of the management layer relies on Fractal components controllers which wrap the administrated legacy software. More precisely, the Fractal component model allows management of:

- **Attribute:** An attribute is a configurable property of a component. The component's interface exposes getter and setter methods for the attributes. A modification of a component attribute is reflected to the legacy software attribute. For instance apache's port is reflected to the legacy Apache configuration file (*httpd.conf*).
- **Binding:** The component's interface exposes methods for controlling bindings between components. The configuration of bindings in the management layer is reflected to the legacy software layer. For instance, by configuring a binding between an apache and a tomcat component in the management layer, an Apache *httpd* may be connected to a Tomcat server to which it delegates dynamic requests. The implementation of this binding configures the *worker.properties* file in the legacy software.
- **Life cycle:** The component's interface exposes methods for controlling the component's execution. The basic lifecycle operations of a legacy system can be performed through this lifecycle interface (e.g. starting and stopping the execution of a component). In the case of apache, it is implemented by calling the Apache commands for starting/stopping a server.

Extending the set of the administration functions provided by Jade is performed implementing additional components controllers. This extension capability provided by the Fractal component model is thus a base to provide an extensible autonomic management infrastructure.

Self-repair and self-optimizing

One important autonomic administration behavior we consider in Jade is self-optimization. Self-optimization is an autonomic behavior which efficiently maximizes resource utilization to meet the end user needs with no human intervention required. Jade aims at autonomously increasing/decreasing the number of replicated resources used by the application when the load increases/decreases. This has the effect of efficiently maximizing resource utilization (i.e. no resource overbooking).

To this purpose, a QoS manager uses sensors to measure the load of the system. These sensors can probe the CPU usage or the response time of application-level requests. The QoS manager also uses actuators to reconfigure the system. Thanks to the generic controller interfaces of Fractal, the actuators used by the QoS manager are themselves generic, since increasing/decreasing the number of resources of an application is implemented as adding/removing components in the application structure.

Besides sensors and actuators, the QoS manager makes use of an analysis/decision component which is responsible for the implementation of the QoS-oriented self-optimization algorithm. This component receives notifications from sensors and, if a reconfiguration is required, it increases the number of resources by allocating new necessary and available nodes. It then deploys those software resources on the new nodes and adds them to the existing application structure, just by creating the associated components on these nodes. Symmetrically, if the resources allocated to an application are under-utilized, the QoS manager performs a reconfiguration to remove some replicas and release their resources.

Another autonomic administration behavior we consider in Jade is self-repair. In a replication-based system, when a replicated resource fails, the service remains available thanks to replication. However, we aim at autonomously repairing the managed system by replacing the failed replica by a new one. Our current aim is to deal with fail-stop faults. The implemented repair policy rebuilds the failed managed system as it was prior to the occurrence of the failure. To this purpose, the failure manager uses sensors that monitor the health of the used resources through probes installed on the nodes hosting the managed system; these probes are implemented using heartbeat techniques. The failure manager also uses a specific component called the *System Representation*. The *System Representation* component maintains a representation of the current architectural structure of the managed system, and is used for failure recovery. One could state that the underlying component model could be used to dynamically introspect the current architecture of the managed system, and use that structure information to recover from failures. But if a node hosting a replica crashes, the component encapsulating that replica is lost; that is why a *System Representation* mechanism is necessary. This representation is reflects the current architectural structure of the system (which may evolve); and is reliable in the sense that it is itself replicated to tolerate faults. The *System Representation* is implemented as a snapshot of the whole component architecture. It is moreover managed through specific components controllers, where each component has a controller in charge of updating its system representation when required.

Besides the system representation, the sensors and the actuators, the failure manager uses an analysis/decision component which implements the autonomic repair behavior. It receives notifications from the heartbeat sensors and, upon a node failure, makes use of the *System Representation* to retrieve the necessary information about the failed node (i.e., software resources that were running on that node prior to the failure and their bindings to other resources). It then allocates a new available node and redeploys those software resources on the new node. The *System Representation* is then automatically updated according to this new configuration.

3. Summary and future work

This paper is a summary of our on-going work on Jade, an environment for implementing autonomic administration software. The main motivation of this project is to deal with the difficult issues that administrators have to face in today's distributed computing environments. The important issues we identified are:

- Reduce the complexity of configuration and deployment
- Limit human intervention for administrating software

We address these issues in Jade by using a component model which provides a unified view of both the managed application and the administration system itself. The component model provides a unified interface which allows for the definition of autonomic behaviors, reducing human interventions. The unified interface is implemented by Fractal controllers, which has permit to separate the different management aspects and combine them in a proper way. The set of standard Fractal component controllers has been extended to take into account some specific management requirements, such as the self-repair aspect. Finally, the ability to extend the set of management aspects by implementing additional Fractal controllers is a basic principle for the provision of an extensible management infrastructure. Our experiments show the potential of this approach. We are currently experimenting with more elaborated autonomic administration services which aim at repairing a wider spectrum of failures (software faults, network partitions …). We are also exploring the definition of an autonomic immune system which would be able to protect the application and the infrastructure.

4. References

[1] E. Bruneton, T. Coupaye, and J.B. Stefani. *Recursive and Dynamic Software Composition with Sharing*. Seventh International Workshop on Component-Oriented Programming (WCOP02), Monday, June 10, 2002, Malaga, Spain. http://fractal.objectweb.org/

[2] J. O. Kephart, D. M. Chess. The Vision of Autonomic Computing. *IEEE Computer Magazine*, Volume 36, Number 1, 2003.

[3] S. Bouchenak, F. Boyer, D. Hagimont, S. Krakowiak, A. Mos, N. de Palma, V. Quéma, J. B. Stefani. Architecture-Based Autonomous Repair Management: An Application to J2EE Clusters. *24th IEEE Symposium on Reliable Distributed Systems (SRDS-2005)*, Orlando, FL, USA, October 2005.

[4] C. Taton, S. Bouchenak, F. Boyer, N. De Palma, D. Hagimont, A. Mos, Self-Manageable Replicated Servers, VLDB Workshop on Design, Implementation, and Deployment of Database Replication, Trondheim, Norway, August 2005.