# Architecture-Based Autonomous Repair Management: Application to J2EE Clusters

S. Bouchenak, F. Boyer, D. Hagimont, S. Krakowiak, N. de Palma, V. Quema, J.-B. Stefani
INRIA Rhône-Alpes, 655 Avenue de l'Europe, Montbonnot, 38334 St Ismier, Cedex France

## Abstract

*This paper presents a component-based architecture for autonomous repair management in distributed systems and its application to J2EE server clusters, called* JADE. *The architecture features three major elements: (1) a dynamically configurable, component-based structure that exploits the reflective features of the* FRACTAL *component model; (2) an explicit and configurable feedback control loop structure, that manifests the relationship between managed system and repair management functions; (3) an original replication structure for the management subsystem itself, which makes it fault-tolerant and self-healing.*

## 1 Introduction

Autonomic computing [5], which aims at the construction of self-managing and self-adapting computer systems, has emerged as an important research agenda in the face of the ever-increasing complexity and pervasiveness of networked computer systems. Following [6], an important part of this agenda lies in the elicitation of architectural principles and design patterns, as well as software engineering techniques for the construction of autonomic systems. As a contribution to this goal, we present in this paper the design of a self-healing failure repair management system, and the application of this design to the construction of an autonomous repair management system for J2EE clusters.

We call *repair management* an instance of failure recovery management, whose main goal is to restore a managed system, after the occurrence of a failure, to an active state satisfying a given level of availability, according to a given policy. For instance, a simple repair policy can be to bring back the failed system to a known configuration which existed prior to failure. Typically, repair management can be used in complement of classical fault-tolerance mechanisms to ensure a managed system satisfies an agreed level of availability.

We have applied our repair management design to build a prototype repair management system for J2EE application server clusters, called JADE. Our J2EE repair management system demonstrates that our architecture can be applied to non-trivial legacy systems, and improves on the state of the art in J2EE cluster management by demonstrating how availability management can be made entirely automatic, even in presence of failures in the management sub-system itself.

## 2 Component basis

Our repair management system is built using the FRACTAL reflective component model [2]. The FRACTAL component model is a reflective component, intended to ease the development of dynamically reconfigurable systems. A FRACTAL component is a run-time entity that is encapsulated, and that has a distinct identity, one or more interfaces (access points) that can be either client interfaces or server interfaces. A system architecture in FRACTAL desccribes two kinds of relationships between components: containment relationships, which describe how components are composed out of other components, and binding relationships, which describe how components communicate. Bindings, or communication paths betwen components, are reified as FRACTAL components.

The originality of the FRACTAL model lies in its reflective structure, that allows to attach different controllers to a component. A controller provides access to a component internals, allowing the internal structure and execution of a component to be externally monitored and controlled. Example controllers in FRACTAL include: interceptors, that allow to intercept incoming and outgoing method invocations at interfaces of a component; attribute controllers that support getter and setter methods for a component's attributes; life-cycle controllers, that allow to manage the execution of a component and the main states it passes through; content controllers that allow to inspect and modify the subcomponents of a composite component.

The FRACTAL component model in used in three main ways: for obtaining a dynamically reconfigurable structure, for instrumenting the managed system, and for building a causally connected system representation.

Proceedings of the Second International Conference on Autonomic Computing (ICAC'05)
0-7695-2276-9/05 $ 20.00 **IEEE**

## 3 Repair management architecture

Our repair management architecture is a FRACTAL software architecture that comprises a managed system, and an explicit feedback control loop built out of: probes and actuators; a manager subsystem; and a transport subsystem, which transfers notifications from sensors to the manager subsystem, and commands from the manager subsystem to actuators.

The *managed system* is defined by a set of components, called nodes, together with their subcomponents. Nodes are abstractions of physical computers. Subcomponents of a node typically correspond to software components executing on a node. A node component provides basic operations for deploying and configuring its subcomponents. For instance, in the JADE system, subcomponents of a node comprise middleware components involved in the different tiers in J2EE server, and J2EE application components.

*Sensors* provide basic monitoring facilities such as node failures or resource usage. *Actuators* provide basic actions or commands, necessary to control the execution of managed components. Sensors are actuators are implemented as FRACTAL controllers of managed components. The transport subsystem binds sensors, actuators, and the manager subsystem. Notifications (e.g. of node or software component failures) follow an asynchronous operation semantics. Commands issued by the manager subsystem obey a synchronous, at most once operation semantics.

The *manager subsystem* is built as a composite component that contains: policy components, including a configuration manager, a repair manager and an administrator console, responsible for implementing the analysis and decision stage of the repair management control loop; and a system representation component (or SR). The SR maintains an abstract view of the system for the benefit of policy components. This abstract view corresponds to a software architecture description of all the components in the system, including control loop components and the manager subcomponents themselves. The consistency between the actual state of the running system and the system representation is maintained by policy components, which update the system representation upon receipt of notifications from sensors, or upon the completion of commands on actuators.

To make the whole system self-healing, the manager component, together with its self-describing SR subcomponent, is in fact replicated. A failure that impacts the manager subsystem can be tolerated, and it can be repaired just as any failure impacting the managed system, following the same or a different policy, as required. The active replication scheme used for the manager relies on a new uniform atomic broadcast protocol, which has been optimized for scalability, throughput and latency in high-performance PC clusters.

## 4 Implementation

The JADE prototype implements the repair management architecture described above. It targets mainly clusters of multi-tier J2EE application servers, but it can be applied in other areas. The implementation relies on the wrapping of legacy components, implementing attribute, binding and life-cycle controllers for components wrapping e.g. the Apache HTTP server, the Tomcat servlet container, the JoNAS EJB server, and the MySQL database server. The system representation is built automatically from the FRACTAL ADL description of the managed system configuration, and provides a run-time view of the system state, as required by the repair management policy components. In particular, interconnections between J2EE middleware tiers, are manifested as FRACTAL bindings.

## 5 Conclusion

Our work draws on, and its related to a wide range of work in cluster systems management such as BioOpera [1], in distributed configuration management, such as [4], or J2EE availability management [3]. It is original in its two-level reflective structure (at component-level and system-wide level), and its systematic component basis. This integrated design has three main benefits: a highly flexible and dynamically reconfigurable structure; the automation of a large part of configuration management activities, making them less error-prone and time-consuming, and the enabling of a self-healing structure.

## References


[1] W. Bausch, C. Pautasso, R Schaeppi, and G. Alonso. BioOpera: Cluster-Aware Computing. In *Proc. IEEE International Conference on Cluster Computing (CLUSTER 2002)*. IEEE Computer Society, 2002.

[2] E. Bruneton, T. Coupaye, M. Leclercq, V. Quéma, and J.B. Stefani. An Open Component Model and its Support in Java. In *Proceedings CBSE '04, LNCS 3054*. Springer, 2004.

[3] G. Candea, S. Kawamoto, Y. Fujiki, G. Friedman, and A. Fox. A Microrebootable System - Design, Implementation, and Evaluation. In *Proceedings OSDI '04*, 2004.

[4] E. M. Dashofy, A. van der Hoek, and R. N. Taylor. Towards architecture-based self-healing systems. In *Proceedings of the 1st Workshop on Self-Healing Systems, WOSS 2002*. ACM, 2002.

[5] J. O. Kephart and D. M. Chess. The Vision of Autonomic Computing. *IEEE Computer 36(1)*, 2003.

[6] S. White, J. Hanon, I. Whalley, D. Chess, and J. Kephart. An Architectural Approach to Autonomic Computing. In *Proceedings ICAC '04*, 2004.




Proceedings of the Second International Conference on Autonomic Computing (ICAC'05)
0-7695-2276-9/05 $ 20.00 **IEEE**