

A Framework for Automating the Performance Management of Component-Based Enterprise Systems

Ada Diaconescu¹

Performance Engineering Laboratory, Dublin City University

Abstract. The presented research abstract describes a framework for automating the performance management of complex, component-based systems. The adopted approach is based on the alternate usage of multiple component variants, with equivalent functional characteristics, each one optimized for a different running environment. A proof-of-concept prototype has been implemented and tested for managing EJB applications running on the JBoss J2EE server.

1 Problem and Motivation

Component technologies [1] such as J2EE [2] and .NET are increasingly being used for building complex enterprise applications. While successfully addressing complex system functionality issues and flexibility requirements, current component technologies provide little support for managing the emerging performance of systems assembled from distinct components. Static component testing and tuning procedures are typically run in isolation or simulated environments. Although important, such procedures provide insufficient performance guarantees for components that are to be run in diverse component assemblies, under unpredictable workloads and on different platforms. Furthermore, the environmental conditions in which a component is initially deployed and run as part of a software application can periodically change during the component's lifetime. Such environmental conditions include the incoming workload and the software and hardware resources available to the considered component. The hardware resources available to a component include memory, CPU and network bandwidth. Software resources can involve other components or applications that the component uses - directly or indirectly - for providing its functionality; such resources include relational databases, the middleware platform, or the JVM.

Changes in a component's running conditions can significantly impact its performance characteristics, including the component's throughput and response times. Often there is no single component implementation or deployment configuration that can yield optimal performance in all possible conditions under which a component may run. This is because optimising a component for all the execution environments that will ever occur throughout the component's lifetime would be prohibitively expensive, if at all possible. Predicting all possible execution contexts in which a component will be deployed and run would be an almost

impossible task. Furthermore, even if this was indeed achieved, implementing all the optimisation solutions - one for each of the predicted running contexts - into one single component would generally not be a good design approach. Such a monolithic component would prove difficult to build and manage, as the implemented business logic would be mingled with the management code used for selecting which optimisation solution to use in each execution environment. Additionally, many of the component's provided solutions would probably not be needed when the component runs as part of a certain application, on a certain platform; different optimisation solutions would be needed when the component runs as part of different applications, on various platforms.

Consequently, because of the increased complexity of current software systems, the performance management of such systems becomes a difficult task, at best. Manually performing the required adaptation and reconfiguration operations, while abiding by system performance and availability constraints, is becoming an ever more costly and risky approach.

2 Background and Related Work

To the best of our knowledge, there are no complete solutions that employ monitoring, evaluation and adaptation for applications based on contextual composition frameworks [1], at the software component level. General frameworks for self-adaptive systems are presented in [3], or [4], featuring inter-related monitoring, analysis and adaptation tiers. Our proposed solution aligns with these approaches, while specifically targeting enterprise applications based on contextual composition middleware [1].

Adaptation techniques based on redundant components, such as presented in [5], are similar, and to some extent complementary, to the application adaptation approach proposed by our research. However, most of these techniques impose various requirements on component providers, such as supplying accurate performance information for each redundant component, at deployment time, or implementing replacement mechanisms for each separate pair of redundant components to be managed. Such requirements may, in our opinion, dramatically increase the cost of implementing such adaptation techniques, or even constrain their applicability. The adaptation decision algorithms defined in [5] can be adopted for our framework when managing the application types for which these algorithms were devised.

Efforts towards standardizing and implementing hot-deployment functionalities in J2EE applications are being made in research initiatives such as [6] and [7]. The presented research will comply with such specified standards and possibly adopt available hot-deployment implementations for the solution implementation.

3 Research Hypothesis

The performance management process of complex software systems is becoming increasingly more costly and error prone [8]. Automating the performance management tasks for such systems becomes an imperative undertaking for successfully overcoming such difficulties. Software systems should consequently be equipped with self-management capabilities, so as to be able to self-optimize in their initial running environment and subsequently adapt to changes in their execution conditions. More precisely, applications should be capable to automatically change their implementation and configuration at runtime, so as to adapt to their running environments and yield optimal performance at all times.

4 Proposed Solution

The presented research describes a solution for automating the performance management of component-based software applications. The adopted approach is based on provisioning multiple component variants with equivalent functional characteristics, each one optimized for a different running environment. These components are referred to as redundant components. As part of the proposed solution, an automatic management framework was devised for dynamically analysing, selecting and activating the redundant components that yield optimal performance in each particular execution environment.

The principal management functionalities provided by the proposed management framework are system monitoring, performance anomaly detection, component evaluation, adaptation decision and component activation [9], [10], [11], [12]. These functionalities work in a feedback-loop manner: the effects of an application adaptation operation are monitored and evaluated and the adaptation decision mechanism accordingly tuned. Thus, the management system can learn and improve its efficiency over time. The framework's main functionalities are briefly discussed over the following paragraphs.

The monitoring functionality is responsible for collecting runtime data from the application components and their execution environment. This data is used for detecting performance anomalies and important variations in the components' execution environments. Monitoring data is also stored and analysed for inferring higher-level information on the performance characteristics of the available redundant components.

The anomaly detection functionality of the framework is responsible for identifying and signalling the occurrence of performance problems, or of relevant variations in the execution environment. Performance anomalies are generally signalled when performance metrics such as response times and throughputs do not meet the system's performance requirements. In addition, the anomaly detection facility identifies and analyses any significant variations that may occur in the execution environment; it then predicts how these variations could potentially affect the system's performance in the immediate future. As a result, the framework may preemptively adapt the application in order to prevent predicted performance problems from actually occurring.

The component evaluation functionality of the framework is responsible for determining the redundant components that are optimal in a given execution environment. The component evaluation process is based on the performance information that exists on the available redundant components, at the time the process is being executed. Initial performance information can be provided based on test results and previous experiences with the considered component [9], [10], [11], [12]. However, such initial information is not compulsory to be provided at component deployment time. The framework can obtain performance information when using its learning abilities. As part of the framework's learning process, performance information is inferred from accurate monitoring data obtained while components are running in the targeted managed system. Any available information is subsequently validated and constantly updated, as new monitoring data becomes available.

The adaptation decision functionality of the framework is responsible for finding optimal solutions to detected or predicted performance problems. Solutions consist of application adaptations that involve the hot-swapping of one or multiple redundant components. The redundant components to be activated are being indicated by the component evaluation facility. In addition to this, the adaptation decision process needs to further judge whether the potential benefits of the considered optimisations would overcome the costs induced by the actual adaptation process. It also needs to ensure that local component optimisations would positively affect the global performance levels of the overall system.

The component activation functionality of the framework is responsible for performing the actual component hot-swapping operations, while maintaining system consistency and the validity of existing user references [9], [10], [11], [12].

The remaining of this section emphasises some of the key features that characterise the proposed management solution:

- Separation of application business logic from management code, increasing system flexibility and manageability
- No extra-requirements imposed on component developers or providers; any standard component can be managed by the proposed framework. Component variants can be acquired from multiple providers and can be dynamically added, updated and/or deleted.
- Support for both centralised and decentralised management approaches. Centralised evaluation and decision mechanisms perform global application adaptation operations for the overall optimisation of the entire application. Local, component-level management operations are also supported for performing more frequent optimisations at reduced costs.
- Decision policies defined for automating various management tasks, at different abstraction levels: specifying system performance goals, adapting and optimising the managed application, inferring component performance information from monitoring data and controlling the correct functionality of the management system itself [10].
- No extra framework implementation or installation efforts required for managing new applications. The framework is implemented once and then used

for managing all applications subsequently deployed and run on a certain application server

- With respect to the system’s functionality, clients remain completely unaware of runtime system management operations.

A proof-of-concept framework prototype was implemented for managing EJB applications running on the JBoss J2EE server [13]. The potential benefits of the presented solution were verified by testing the principal framework functionalities on several sample applications, such as the Duke’s Bank [14]. Future work will focus on extending the policy-based evaluation and decision functionalities and test the overall framework under further example scenarios.

References

1. Szyperski, C., et al: Component Software: Beyond Object-Oriented Programming. Addison-Wesley, Great Britain (November 2002)
2. Sun Microsystems: (The Java 2 Platform, Enterprise Edition technology (J2EE)) <http://java.sun.com/j2ee/>.
3. Cheng, S., et al: Rainbow: Architecture-bases self adaptation with reusable infrastructure. *IEEE Computer* **37** (2004)
4. Oriезy, P., et al.: An architecture-based approach to self-adaptive software. *IEEE Intelligent Systems* (1999)
5. Yellin, D.M.: Competitive algorithms for the dynamic selection of component implementations. *IBM Systems Journal* **42** (2003)
6. Sun Microsystems: JSR88: J2EE Deployment API Specification. (2003) <http://java.sun.com/j2ee/tools/deployment/>.
7. Matevska-Meyer, J., Olliges, S., Hasselbring, W.: Runtime reconfiguration of j2ee applications. (In: DECOR 2004) 77–84
8. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. *IEEE Computer* (2003)
9. Diaconescu, A., Mos, A., Murphy, J.: Automatic performance management in component-based software systems. In: Proceedings of the International Conference on Autonomic Computing (ICAC04), New York, USA (May 2004) 214–221
10. Diaconescu, A., Murphy, J.: A framework for automatic performance monitoring, analysis and optimization of component based software systems. In: Workshop on Remote Analysis and Measurement of Software Systems (RAMSS), International Conference on Software Engineering (ICSE 2004), (Edinburgh, Scotland, UK)
11. Diaconescu, A., Murphy, J.: A framework for using component redundancy for self-adapting and self-optimising component-based enterprise systems. In: The ACM SIGPLAN Student Research Competition (3rd place), International Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA 2003), Anaheim, California, USA (2003) 390–391
12. Diaconescu, A., Murphy, J.: A framework for using component redundancy for self-optimising and self-healing component based systems. In: Workshop on Software Architectures for Dependable Systems (WADS), International Conference on Software Engineering (ICSE2003), Hilton Portland, Oregon USA (2003)
13. JBoss: (JBoss J2EE application server) www.jboss.org.
14. Sun Microsystems: (Duke’s Bank - a sample J2EE application) http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/Ebank.html.