Proceedings of the 6th International Conference on
Autonomic Computing and Communications, Barcelona,
Spain, June 2009.

# Applying Adaptation Design Patterns *

Andres J. Ramirez and Betty H.C. Cheng
Michigan State University
3115 Engineering Building
East Lansing, Michigan 48824
{ramir105, chengb}@cse.msu.edu

## ABSTRACT

Dynamic adaptation may be used to prevent software downtime while new requirements and responses to environmental conditions are incorporated into the system. Previously, we studied over thirty adaptation-related projects to develop twelve adaptation-oriented design patterns that can be leveraged from one adaptive system to another. This paper presents a case study in which we apply our adaptation patterns in the design of a dynamically adaptive news web server. This pattern-based design separates the functional logic from the adaptive logic, resulting in a system that supports dynamic adaptation and is easier to maintain and analyze. Furthermore, to address assurance concerns, we applied automated formal verification techniques to ensure instantiated pattern models satisfy invariant properties specified in each adaptation pattern.

## Categories and Subject Descriptors

D.2.11 [**Software Architectures**]: Design Patterns; D.2.10 [**Design**]: Methodologies

## General Terms

Design.

## Keywords

Design Patterns, Adaptive Systems, Autonomic Systems

## 1. INTRODUCTION

Dynamic adaptation may be used to prevent costly software downtime while new requirements and responses to environmental conditions are incorporated into the system. A current lack of reusable design expertise that can be leveraged from one adaptive system to another complicates the design, verification, and validation of adaptive and autonomic systems. Previously, we studied over thirty adaptation-related projects to develop adaptation-oriented design patterns [3]. Each adaptation pattern comprises key fields that specify when to apply the pattern, captures the generalized solution through UML diagrams, gives templates for specifying constraints that must be satisfied by the instantiated models, and enumerates potential consequences as a result of applying the pattern. These adaptation patterns can be categorized according to their main objective as monitoring, decision-making, and reconfiguration. Moreover, each adaptation pattern can be selectively combined with other design patterns as required.

This paper overviews ZAP.com, a re-engineered version of Z.com, the adaptive news web server originally presented by Garlan *et al.* [1]. Both applications were designed to address the "slashdotting effect" where news sites that are widely publicized are unable to handle the large number of content requests, and either suffer from high latency or are unable to serve content altogether. We built ZAP.com from scratch by instantiating combinations of our monitoring, decision-making, and reconfiguration design patterns. Specifically, our monitoring patterns provide an infrastructure to observe both an application and its environment. Similarly, our decision-making patterns focus on interpreting monitoring data and selecting reconfiguration plans when appropriate. Lastly, our reconfiguration patterns specify the steps required to safely reconfigure the application at run time.

From this case study, we see that our collection of adaptation patterns provide a valuable resource for developers seeking to leverage adaptive design experience. For instance, our patterns facilitate the development of systems amenable to change by separating the functional logic from the adaptive logic. Similarly, developers can select and instantiate adaptation patterns that will be required at run time, potentially avoiding applications bloated with adaptive functionality that may not be required. Moreover, to address assurance concerns, developers can leverage automated formal verification tools, such as Hydra [2], to ensure instantiated models satisfy the properties specified in each pattern.

## 2. EXAMPLE INSTANTIATION

ZAP.com is modeled as a set of clients and servers with the overall constraint that latency must fall within a given threshold. We developed ZAP.com in three major stages, closely following the model-based development process for adaptive systems [4]. First, we modeled and implemented the business logic according to the local properties and functional requirements identified for Z.com [1]. Specifically, we designed ZAP.com as an object-oriented multi-threaded server-client architecture capable of accepting incoming HTML requests, determining the current system workload, and redirecting requests in order to balance the overall system workload.

Next, the functional logic of ZAP.com must be augmented with a monitoring, decision-making, and reconfiguration infrastructure to enable the system to dynamically adapt to

*This work has been supported in part by NSF grants CCF-0541131, CNS-0551622, CCF-0750787, IIP-0700329, and CCF-0820220, Army Research Office W911NF-08-1-0495, and the Department of the Navy, Office of Naval Research under Grant No. N00014-01-1-0744, Ford Motor Company, and a grant from Michigan State University's Quality Fund.


*ICAC'09,* June 15–19, 2009, Barcelona, Spain.
ACM 978-1-60558-564-2/09/06.

new requirements and environmental conditions. For the second development stage, we selected and instantiated a set of monitoring and decision-making patterns and integrated them with the functional logic of ZAP.com (see Figure 1). Specifically, the monitoring and decision-making patterns are responsible for probing the latency of servers, identifying abnormal behavior, and selecting appropriate reconfiguration plans that best balance costs, latency, and fidelity constraints given its current environment.

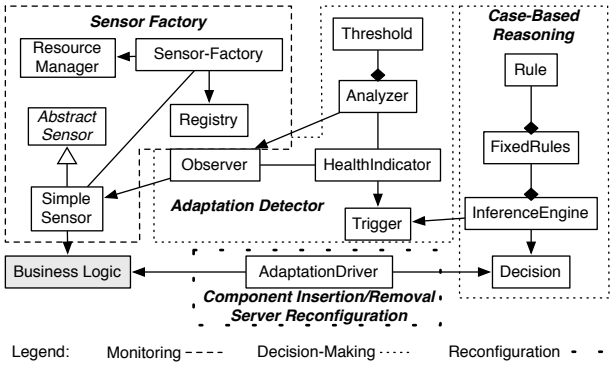


**Figure 1: Structural model for ZAP.com.**

We selected the Sensor Factory monitoring pattern to periodically probe the average latency of servers for two reasons. First, ZAP.com is susceptible to periods of high workloads. As a result, ZAP.com requires a distributed monitoring scheme that minimizes the impact of constantly probing servers. Second, the business logic already provided an interface to the desired monitoring attributes, thus simplifying interactions between sensors and servers. After we instantiated the models, we invoked the Hydra tool [2] to automatically translate state-based models into Promela code. At this step, we corrected a constraint violation in the instantiated models that enabled potential inconsistencies in multiple distributed resource managers.

The first decision-making pattern we applied to ZAP.com was Adaptation Detector. This decision-making pattern was selected to process the monitoring data obtained from sensors deployed by the Sensor Factory pattern and detect when a reconfiguration was required. In particular, numeric thresholds establish boundaries of normal or expected behaviors. If a monitoring data feed exceeds these thresholds, then it is likely that a reconfiguration is required. After instantiating this pattern, we used the Hydra tool [2] to ensure no properties specified in the pattern template were violated.

Next, we applied the Case-based Reasoning decision-making pattern to dynamically select a reconfiguration plan based on monitoring information. Adaptation concerns for ZAP.com are multi-faceted and include balancing costs, latency, and fidelity constraints. We were able to succinctly encode our reconfiguration plans as a set of "if-then-else" rules based on the adaptive requirements. Specifically, ZAP.com comprises a collection of rules that map incoming monitoring events, in this case from the Adaptation Detector, with specific reconfiguration plans (see Figure 1). Finally, we used Hydra [2] to ensure our models satisfied properties listed in the pattern template.

At this stage, we have augmented ZAP.com with monitoring and decision-making capabilities. Specifically, we can periodically observe the average latency of servers, detect when a reconfiguration is required, and select a reconfiguration plan that yields the desired behavior. For the third development stage, we introduced a reconfiguration infrastructure that leveraged three of our reconfiguration patterns to address the required steps for safely reconfiguring servers at run time. While two reconfigurations deal solely with parameter tuning, namely switching the content delivery mode, the other possible reconfigurations involve adding or removing servers at run time. We applied the Component Insertion, Component Removal, and Server Reconfiguration patterns to ensure these reconfigurations do not leave ZAP.com in an inconsistent state.

The Component Insertion and Component Removal reconfiguration patterns explicitly manage the operational states of ZAP.com components. These patterns guide any component affected by the reconfiguration to a safe state before the reconfiguration is applied. Components that reach a safe state may not initiate new transactions until notified by the reconfiguration infrastructure. As a result, ZAP.com is guaranteed to be in a consistent state before, during, and after a reconfiguration. Moreover, the Server Reconfiguration applied both the Component Insertion and Component Removal patterns to safely add or remove servers at run time. To ensure no client request is lost during a reconfiguration, the Server Reconfiguration pattern queues incoming client requests so these may be processed once the reconfiguration is complete.

# 3. DISCUSSION

For this case study, we augmented ZAP.com's business logic with monitoring, decision-making, and dynamic reconfiguration capabilities by applying our adaptation patterns. Since the resulting design separates the functional logic from the adaptive logic, it is possible to extend the functionality of ZAP.com or to substitute our patterns without severely impacting the overall application. Furthermore, we can assure that ZAP.com will not reach an inconsistent state as a result of a reconfiguration by verifying the instantiated patterns satisfy specific safety and liveness properties. We also note that, in general, developers should leverage adaptation-oriented frameworks [1] when the application's context is compatible with the framework's services. Nonetheless, if the application's context and adaptation needs differ significantly from the existing adaptation-oriented frameworks, then our adaptation patterns may provide a more flexible development alternative.

# 4. REFERENCES


[1] S.-W. Cheng, D. Garlan, and B. Schmerl. Architecture-based Self-Adaptation in the Presence of Multiple Objectives. In *Proceedings of the 2006 International Workshop on Self-adaptation and Self-Managing Systems*, pages 2–8, Shanghai, China, 2006. ACM.

[2] W. E. McUmber and Betty H.C. Cheng. A General Framework for Formalizing UML with Formal Languages. In *ICSE'01: Proc. of the 23rd Intl. Conf. on Soft. Eng.*, pages 433–422, Washington, DC, USA, 2001. IEEE Computer Society Press.

[3] A. J. Ramirez and B. H. C. Cheng. Developing and Applying Design Patterns for Dynamically Adaptive Systems. Technical Report MSU-CSE-09-8, Computer Science and Engineering, Michigan State University, East Lansing, Michigan, March 2009.

[4] J. Zhang and Betty H.C. Cheng. Model-Based Development of Dynamically Adaptive Software. In *Proceedings of the 28th International Conference on Software Engineering*, pages 371–380, Shanghai, China, 2006. ACM.