Rainbow: Architecture-based Self-adaptation with Reusable Infrastructure

Shang-Wen Cheng An-Cheng Huang David Garlan Bradley Schmerl Peter Steenkiste
School of Computer Science, Carnegie Mellon University
{zensoul, pach, garlan, schmerl, prs}@cs.cmu.edu

Abstract

Software-based systems today are increasingly expected to dynamically self-adapt to accommodate resource variability, changing user needs, and system faults. Recent work uses closed-loop control based on external models to monitor and adapt system behavior at run time. Taking this externalized approach, the Rainbow framework we have developed uses software architectural models to dynamically monitor and adapt a running system. A key goal and primary challenge of this framework is to support the reuse of adaptation strategies and infrastructure across different systems. In this poster, we show that the separation of a generic adaptation infrastructure from system-specific adaptation knowledge makes this reuse possible.

1. Introduction

Software-based systems today increasingly operate in changing environments with variable user needs, resulting in a continued rise of administrative overhead for managing these systems. Thus, systems are increasingly expected to dynamically self-adapt to accommodate resource variability, changing user needs, and system faults. Recent work uses a closed-loop control based on external models and mechanisms to monitor and adapt system behavior at run time to achieve various goals [2, 5, 4, 6, 8].

We have developed the Rainbow framework that uses externalized control mechanisms based on architectural models to dynamically monitor and adapt a running system, often at a fairly global, module level. In principle, externalized control mechanisms separate the concerns of functionality from the concerns of "exceptional behaviors," providing several benefits, including analysis, modularity, applicability to legacy systems, and reuse. In previous work ([1]), we have demonstrated the feasibility of Rainbow in dynamically adapting target systems. One of the promises of Rainbow is to provide a low-cost approach for adding self-adaptation capabilities to a wide range of systems. Achieving this generality is the primary challenge we address in this research. We briefly introduce the reusable framework here but describe the two case study systems in the poster.

2. The Rainbow Framework

Figure 1 shows the control loop of the Rainbow framework for self-adaptation. Rainbow monitors the run-time properties of an executing system in the system-layer through an abstract model maintained by the model manager in the architecture layer. The constraint evaluator checks the model for constraint violations and triggers the adaptation engine if a problem occurs. The adaptation engine and the executor then perform adaptations on the running system. The architecture and system layers interact through a translation layer.

2.1. Reusable Units of Rainbow

To fulfill Rainbow's objective, components of the framework need to be reusable across systems. Moreover, the supporting architectural description language and development toolsets should be reusable as well. We divide the framework into an *adaptation infrastructure* and the *system-specific adaptation knowledge*. The adaptation infrastructure provides common functionalities across self-adapting systems and are therefore reusable while the adaptation knowledge may be more difficult to reuse.

System-layer infrastructure. At the system layer, we have defined the necessary system access interface and built an infrastructure that implements the interface. A system measurement mechanism, realized as probes, observes and measures various states of the system. The system information may be published by or queried from the probes. A resource discovery mechanism can be queried for new resources based on resource type and other criteria. An effector mechanism carries out the actual system modification.

Architecture-layer infrastructure. At the architecture layer, *gauges* aggregate information from the probes and update the appropriate properties in the model. A model manager manages and provides access to the architectural model of the system. A constraint evaluator checks the model periodically and triggers adaptation if a constraint violation occurs. An adaptation engine determines the course of action and carries out the necessary adaptation.

Proceedings of the International Conference on Autonomic Computing (ICAC'04)
0-7695-2114-2/04 $20.00 © 2004 IEEE
IEEE COMPUTER SOCIETY

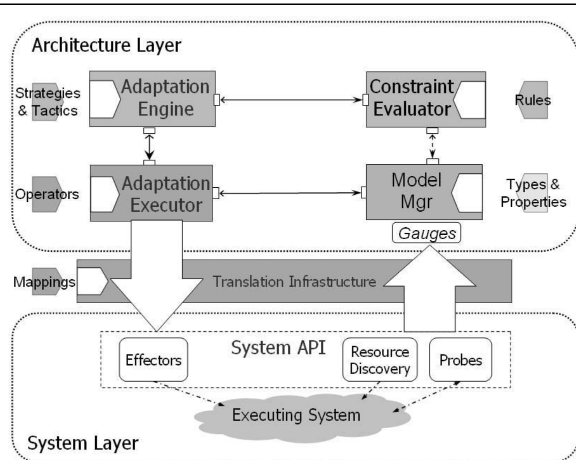

Figure 1. The Rainbow framework.

Translation infrastructure. A translation infrastructure helps to mediate the mapping of information across the abstraction gap from the system to the model and vice versa. A translation repository within the infrastructure maintains various mappings shared by the translator components to, for example, translate an architectural-level element identifier into an IP address or an architectural-level change operator into some system-level operations.

2.2. System-specific Adaptation Knowledge

To add self-adaptation to a system, the adaptation infrastructure needs to be tailored using the system-specific adaptation knowledge, which includes the types and properties of components, behavioral constraints, and strategies for adaptation. A significant amount of the knowledge needs to be specified to apply the framework to different systems, but reuse of this knowledge across systems in similar domains is possible using the notion of an *architectural style*.

An architectural style characterizes a family of systems related by shared structural and semantic properties, and typically provides four sets of things [7]—types, rules, properties, and analyses. Supporting run-time adaptation also requires capturing the dynamic attributes of the system, both in terms of what operations can be done on the system, as well as how the operations can be applied. Therefore, we augment the notion with operators (actions that may be performed on elements) and strategies (adaptation prescription specified in terms of operators) for adaptation.

3. Case Studies on Reuse

To validate our approach, we use two case study systems that have different architectural styles but share the same system concern of performance. The first one consists of a set of Web clients, each of which requests contents from one of several groups of Web servers, and the primary concern is the response time experienced by the clients. The second case study is a video conferencing system that accommodates participants with different devices and conferencing applications, and the primary concern is the available bandwidth between participants.

The three layers of the adaptation infrastructure are reused across the two case studies. Our system-layer infrastructure provides the effector, probing, and resource discovery mechanisms. At the architecture layer, the gauges, model manager, constraint evaluator, and adaptation engine are reused. The two systems also share the translation repository and translators. On the other hand, the two case studies are both concerned with maintaining performance, which is manifested in the properties of their styles. Therefore, the knowledge about the shared properties (e.g., mapping and aggregation knowledge) can be reused across the two systems. Using code size as an approximate measure of reuse, roughly 98% of the 105 kilo-lines of code in the implementation was reused.

Acknowledgments

The research described here was supported by DARPA, under Grants N66001-99-2-8918 and F30602-00-2-0616. Views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of DARPA.

References

[1] S.-W. Cheng et al. Using architectural style as a basis for self-repair. In *Software Architecture: System Design, Development, and Maintenance*. KAP, 2002.

[2] E. M. Dashofy, A. van der Hoek, and R. N. Taylor. Towards architecture-based self-healing systems. In Garlan et al. [3].

[3] D. Garlan, J. Kramer, and A. Wolf, editors. *Proceedings of the First ACM SIGSOFT Workshop on Self-Healing Systems (WOSS '02)*. ACM Press, 2002.

[4] I. Georgiadis, J. Magee, and J. Kramer. Self-organizing software architectures for distributed systems. In Garlan et al. [3].

[5] M. M. Gorlick and R. R. Razouk. Using weaves for software construction and analysis. In *13th International Conference of Software Engineering*, pages 23–34. IEEE, May 1991.

[6] J. Magee, N. Dulay, S. Eisenbach, and J. Kramer. Specifying Distributed Software Architectures. In W. Schafer and P. Botella, editors, *Proceedings of 5th European Software Engineering Conf. (ESEC 95)*, 1995.

[7] R. T. Monroe, A. Kompanek, R. Melton, and D. Garlan. Architectural styles, design patterns, and objects. *IEEE Software*, 14(1):43–52, Jan–Feb 1997.

[8] P. Oreizy, N. Medvidovic, and R. N. Taylor. Architecture-based runtime software evolution. In *20th International Conference of Software Engineering*. IEEE, 1998.

IEEE COMPUTER SOCIETY