5-2009

RAIDE for Engineering Architecture-Based Self-Adaptive Systems

Shang-Wen Cheng
Carnegie Mellon University

David Garlan
Carnegie Mellon University

Bradley Schmerl
Carnegie Mellon University

Published In
Software Engineering - Companion Volume, 2009. ICSE-Companion 2009. 31st International Conference on, 435-436.

RAIDE for Engineering Architecture-Based Self-Adaptive Systems

Shang-Wen Cheng, David Garlan, and Bradley Schmerl
Carnegie Mellon University
{zensoul,garlan,schmerl}@cs.cmu.edu

Abstract

Rainbow is an approach for engineering self-adaptive systems, with run-time, closed-loop control over target systems to monitor, detect, decide, and act on opportunities for system improvement. RAIDE enables adaptation engineers to customize the Rainbow framework, simulate adaptation behavior, and deploy Rainbow run-time components.

1. Introduction

Imagine a world where a software engineer could take an existing software system and specify objectives, conditions for change, and strategies for adaptation to make that system self-adaptive where it was not before. Imagine that this could be achieved in days to weeks, rather than months, while maintaining business goals and other properties of interest. Imagine further that an engineer could reuse and share adaptation expertise and quickly apply others' strategies to her own system.

Increasingly, systems must have the requirement to self-adapt with minimal human oversight. They must cope with variable resources, system errors, and changing user priorities, while maintaining as best as they can the goals and properties envisioned by the engineers and expected by the users. However, self-adaptation in today's systems is costly to build, often taking many man-months to develop or to retrofit systems with the capabilities. Moreover, once added, the capabilities are difficult to modify and usually provide only localized treatment of system errors.

Broad speaking, several dynamic, architecture-based adaptation approaches have been developed to date [2],[5], focusing on formal models, mechanisms of adaptation, or control distribution and decentralization. However, only a few explicitly tackle the engineering challenges of making a target system self-adaptive with an integrated tool, such as ArchStudio [4], Plastik [1], and IBM's Autonomic Computing Toolkit [6].

Our approach, called Rainbow [3], makes it possible for engineers to easily define adaptation policies that are more global in nature (via an architecture model that reflects properties of the executing system) and take into consideration business goals and quality attributes (using utility theory to inform trade-offs). Rainbow enables engineers to augment existing systems to be self-adaptive without needing to rewrite them from scratch, to reuse adaptation policies across similar systems, to synergistically combine multiple sources of adaptation expertise, and to do so in ways that support maintainability, evolution, and analysis.

In our research demonstration, we present a general and customizable framework with an Eclipse-based development environment that supports our adaptation engineering process to make a system self-adaptive.

2. Rainbow adaptation engineering process

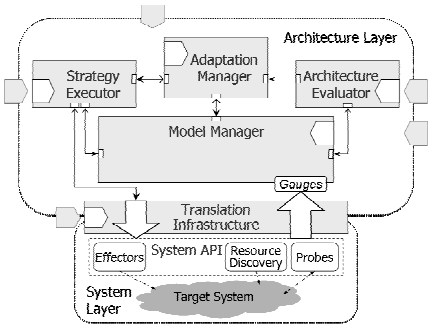


Figure 1. The Rainbow framework

The Rainbow approach consists of a framework, an adaptation language (not presented here), and an adaptation engineering process. The framework (see Figure 1) monitors a target system and reasons about appropriate adaptations to the system using its architecture model. Abstractly speaking, monitoring mechanisms – probes and gauges – observe the running system to update properties of the model managed by the *Model Manager*. *Architecture Evaluator* checks that the system is operating within acceptable bounds, as defined by architectural constraints. Upon constraint violation, *Adaptation Manager* selects the best strategy given current system conditions reflected in the model.

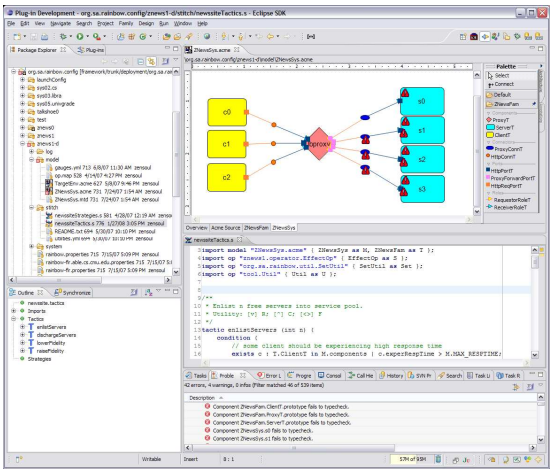
Strategy Executor executes the strategy on the running system via system-level effectors. The cycle repeats.

The *adaptation engineering process* prescribes how to gather the necessary artifacts to tailor the Rainbow framework to a target system. Table 1 summarizes the customization artifacts. Rainbow supports a number of adaptation concepts to allow engineers to focus on adaptation-level concerns: strategies, tactics, operators, effectors, gauges, and probes. A team of *adaptation engineers* – composed of the adaptation integrator, system adapter, style writer, gauge writer, and strategy writer – carries out the iterative process to evolve and augment a system with self-adaptive capabilities.

Table 1. Framework customization points

Framework Component	Customization Artifacts
Translation Infrastructure	Probes, gauges, effectors; Translation mappings
Model Manager	Architecture & Env. models
Architecture Evaluator	Architectural constraints
Adaptation Manager	Adaptation strategies; Utility preference spec.
Strategy Executor	Style operators

3. The Rainbow Adaptation IDE



To support adaptation engineering, we integrated a number of existing tools into a single, Rainbow Adaptation Integrated Development Environment (RAIDE). RAIDE enables the adaptation engineer to edit the Acme architecture models, write Stitch scripts, compose customization artifacts, simulate adaptation behavior, and deploy the Rainbow run time. A Control Console provides central management of the run-time infrastructure and allows control of the distributed delegates, updates to software, testing of effectors, and deploying and undeploying of probes. Rainbow also supports dynamic update of customization artifacts, including utility preferences and adaptation strategies.

RAIDE features the following: (1) A navigator enables managing the customization artifacts; (2) an AcmeStudio plug-in enables visualizing and editing the target-system architecture model, including the system properties to monitor and the architectural constraints to evaluate; (3) a YaML plug-in enables editing the specification files for probes, gauges, and effectors; (4) a Stitch editor enables composing adaptation strategies, with syntax highlighting, simple code completion, on-save parsing, and an outline view; (5) a Utility editor enables specifying preferences and profiles that capture business objectives; (6) a configuration editor enables overall integration of a customized Rainbow instance; and (7) an SDK enables simulating Rainbow adaptation.

To demonstrate low customization effort with Rainbow, we tracked detailed customization activities in two case studies [3]. To summarize, customization took 40 man-hours in one case and 93 man-hours in the other. Note that while the majority of the effort was spent developing monitoring capabilities, the resulting probes and gauges are reusable artifacts. More significant are the orders-of-magnitude of effort: most activities required on the order of minutes to a couple hours, not days, while incremental changes required on the order of tens of minutes, not hours.

In future work, we aim to simplify customization and reduce efforts further using templates, wizards, and default configurations. We would enhance the Stitch editor with code navigation, incremental build, linkage to the architecture and model operators, and adaptation debugging. We would also enhance the Console with finer-grained control of each Rainbow component.

4. References

[1] T.V. Batista, A. Joolia, and G. Coulson. Managing dynamic reconfiguration in component-based systems. In EWSA, LNCS 3527:1-17, Springer, June 13-14, 2005.

[2] J.S. Bradbury, J.R. Cordy, J. Dingel, and M. Wermelinger. A survey of self-management in dynamic software architecture specifications. In Proc. WOSS '04, pp. 28-33, ACM, New York, NY, 2004.

[3] S-W. Cheng. Rainbow: cost-effective software architecture-based self-adaptation. Technical Report CMU-ISR-08-113, Carnegie Mellon U School of CS, May 2008.

[4] E.M. Dashofy, A. van der Hoek, and R.N. Taylor. Towards architecture-based self-healing systems. In Proc. WOSS '02, pp. 21-26, ACM, New York, NY, 2002.

[5] D. Ghosh, R. Sharman, H.R. Rao, and S. Upadhyaya. Self-healing systems—survey and synthesis. Decision Support Systems, 42(4):2164-2185, 2007.

[6] IBM developerWorks. Autonomic computing toolkit. http://www.ibm.com/developerworks/autonomic/overview.html, 2008. [Online; accessed 2-April-2008].