Knowledge-Based Architectural Adaptation Management for Self-Adaptive Systems

John C. Georgas

Institute for Software Research
University of California, Irvine
Irvine, CA 92697-3425
+1 949 824 5160

jgeorgas@ics.uci.edu

ABSTRACT

Self-adaptive software continually evaluates and modifies its own behavior to meet changing demands. One of the main elements in constructing architecture-based self-adaptive software is adaptation policy specification. In this paper, we present an approach to the construction of self-adaptive software based on the architecture-centric definition of knowledge-based adaptation policies. Our approach explicitly represents adaptation policy at the architectural level using techniques inspired from the field of artificial intelligence, and provides for the decoupling of adaptation policy from architectural compositions and for the dynamic runtime evolution of these policies: we believe that this offers key benefits in terms of reuse potential and flexibility. We elaborate on this approach, and discuss our planned evaluation methodology and specific research contributions.

Categories and Subject Descriptors

D.2.11 **Software Engineering**: Software Architectures – *languages*.

General Terms

Management, Design, Languages

Keywords

Architectural adaptation management, self-adaptive software.

Dissertation Advisor

Richard N. Taylor, taylor@ics.uci.edu.

Research Area

Architecture-based self-adaptive software.

Research Topic

The definition of self-adaptive behavior through knowledge-based management of architecture-centric adaptation policies that are strongly decoupled from system implementations and dynamically modifiable at runtime.

1. INTRODUCTION

Self-adaptive software is aimed at addressing the challenges of constructing systems that have the ability to autonomously respond to a variety of situations. *Architecture-based* self-adaptive software focuses on using software architecture models as the central abstraction for decision-making and change enactment in self-adaptive systems; the focus on explicit architectural models addresses the challenges and complications imposed by the multiple artifacts and levels of abstraction through which autonomous behavior can be expressed. A variety of research efforts have adopted this architecture-centric approach including contributions based on dynamic software architecture description languages (ADLs) as well as work based on dynamic distributed systems with explicit architectural models.

A core element in architecture-based self-adaptive systems is the expression of *adaptation policy*: the mapping between the set of conditions indicating the need for an adaptation and the collection of modifications through which this need can be addressed. Adaptation policies encapsulate the timing and specifics of change, addressing when and what needs to be done in order for a system to successfully adapt. These policies, however, are often too closely coupled with the system to which they relate and remain static during runtime; in a domain expressly geared towards creating agile and adaptable systems, these specific shortcomings result in the specification of adaptive behavior that is conceptually hard to reuse across multiple systems and difficult to dynamically modify during system deployment.

Our goal is to address these difficulties by developing mechanisms and supporting infrastructure for the specification of adaptation policies that are strongly decoupled from systems and dynamically modifiable at any point in the system's lifecycle, especially runtime. To achieve this, we propose an architecture-centric and knowledge-based approach in which adaptive behavior is achieved through a body of *observations* establishing known information about a system and a collection of rule-based *adaptation policy* specifications. Explicitly represented and dynamically and independently managed at the architectural level, these artifacts provide for a clear separation of concerns between a system's elementary components and the specification of the policies which define self-adaptive behavior.

The key elements of this approach are the treatment of both adaptation policy and related knowledge as first-class architectural elements explicitly represented as decoupled parts of a software architecture, the dynamic management of policies at runtime, and the integration of established artificial intelligence methods in software engineering research.

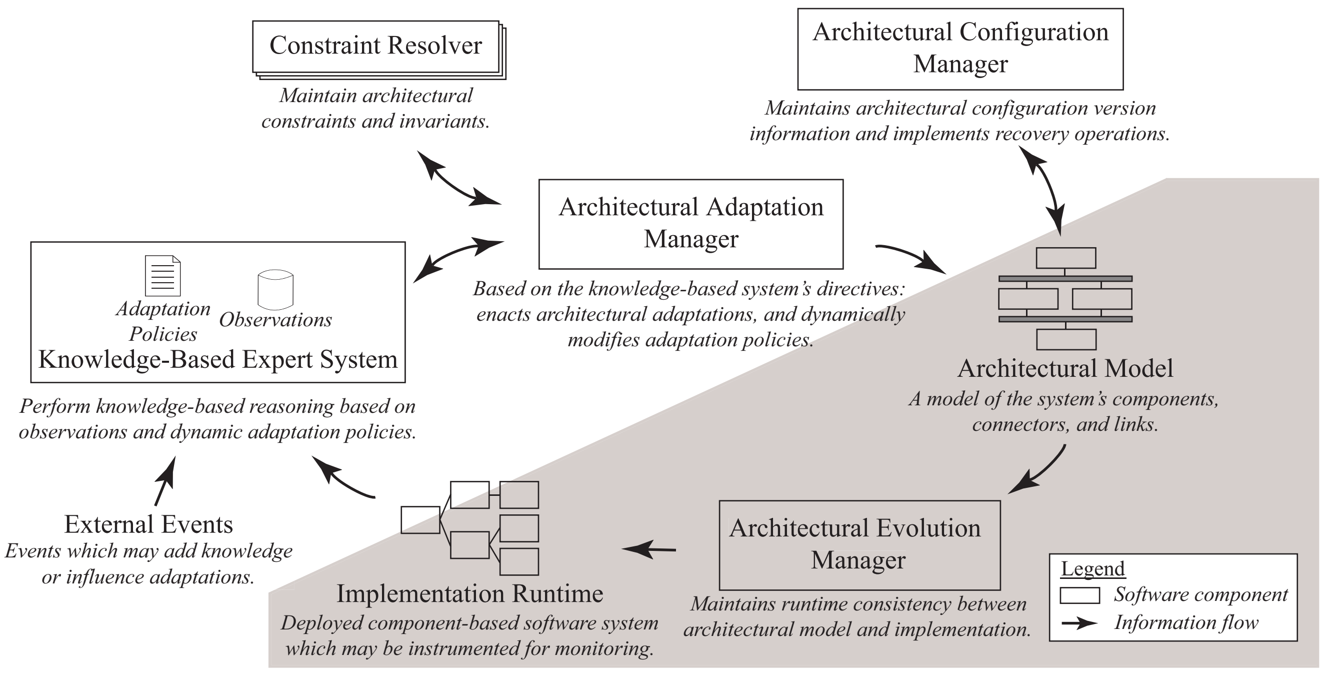
Constraint Resolver
Maintain architectural constraints and invariants.
Architectural Configuration Manager
Maintains architectural configuration version information and implements recovery operations.
Architectural Adaptation Manager
Based on the knowledge-based system's directives: enacts architectural adaptations, and dynamically modifies adaptation policies.
Adaptation Policies
Observations
Knowledge-Based Expert System
Perform knowledge-based reasoning based on observations and dynamic adaptation policies.
Architectural Model
A model of the system's components, connectors, and links.
External Events
Events which may add knowledge or influence adaptations.
Implementation Runtime
Deployed component-based software system which may be instrumented for monitoring.
Architectural Evolution Manager
Maintains runtime consistency between architectural model and implementation.
Legend
Software component
Information flow

Figure 1. A high-level overview of our proposed approach, indicating the role of knowledge-based policy specification—the area in which our work is primarily focused—in the larger context of adaptation management. Shaded areas illustrate existing evolution management facilities we leverage.

The following sections outline background information and a justification for the need of this research direction in Section 2, our hypothesis and proposed approach in Section 3. Our expected contributions and planned evaluation activities appear in Section 4 and Section 5 respectively.

2. BACKGROUND AND JUSTIFICATION

A formulation of the processes involved in architecture-based self-adaptive systems appears in [12]. In this formulation, the process of self-adaptation is separated into two sub-processes: *evolution* management, which is focused on the enactment of changes centered on explicit architectural models, and *adaptation* management, concerned with the decision-making process guiding these changes. While specific methods and technologies are presented for evolution management, the adaptation management process is not similarly addressed. In our research, we expand on this approach to provide for specific representations, methods, techniques, and tools for adaptation management.

Dynamic software architecture specification research has also addressed the issue of architecture-based adaptation management and the specification of adaptation policy. A variety of specification and reconfiguration languages, such as Darwin [10], Wright [3], Gerel [7], and Acme style-based specifications [8], provide facilities for the definition of adaptation policy; however, these definitions are coupled either with specific architectural elements or specific architectural compositions and do not exhibit runtime dynamism. Other notations and approaches, such as COMMUNITY [16] and CHAM [15], provide for varying degrees of adaptation policy decoupling but do not explicitly address the runtime dynamism of these policy specifications. More application-oriented approaches such as self-adaptive digital signal processing systems [14] and coordination-based object-oriented software [4] exhibit similar drawbacks. The K-Components approach [6] to distributed systems provides support for decoupled and dynamic policy specification, but these policies are limited to rudimentary component replacement operations.

A survey of these adaptation policy definition techniques in architecture-based self-adaptive software research indicates the need for an approach to policy specification which is strongly decoupled from both the architectural elements and the topologies they relate to, in addition to providing for the dynamic management of these policies at system runtime.

3. HYPOTHESIS AND APPROACH

We hypothesize that the above outlined drawbacks of existing architecture-centric adaptation policy specifications can be effectively addressed by adopting a *knowledge-based* approach in which self-adaptive behavior is specified through a collection of independent and dynamic *rule-based* policy definitions which reason over a body of architectural *observations* encapsulating knowledge about a system. To promote visibility and understandability, both of these constructs must be explicitly represented, expressed, and manipulated at the architectural level. Additionally, these rule-based policies must be independent and decoupled from the architectural building blocks used to construct a self-adaptive system and be dynamically modifiable during system runtime. With this approach, we enable systems that can not only autonomously adapt, but also change the manner in which they do so.

An overview of the encompassing approach we envision appears in Figure 1; this overall process contextualizes our work on dynamic adaptation policy specification within the larger process of adaptation management. The *Architectural Evolution Manager* (AEM) [5] explicitly manages consistency between architectural models and the systems which reify them; adaptations to systems, then, can be expressed solely as changes to their architectural models. Information gathered about the running system is collected in an *adaptation knowledge-base* and forms a body of *observations* over which rule-based *adaptation policies* reason. The outputs of this knowledge-base, which is managed at runtime by an expert system, are *adaptation responses* deemed necessary by policy specifications and known information; these responses primarily describe modifications to architectural structure but may also apply to the observations and policies encapsulated in the knowledge-base. If these responses are not prohibited by a collection of *constraint resolvers*, they are applied to the architectural model of the self-adaptive system being managed; changes are recorded by an Architectural Configuration Manager, which may be used to manually recover from undesirable changes. Finally, the AEM is responsible for the actual enactment of these changes to runtime systems, at which point the cyclic process resumes.

3.1 Adaptation Policies

Our primary focus is the development of decoupled and independently dynamic adaptation policies which are modeled at the architectural level as first-class entities (by this, we mean entities which can be explicitly referenced and manipulated rather than being subsumed by other architectural constructs).

Each adaptation policy is defined through a set of composable rule-based policy specifications, capturing the logic and goals of a particular adaptive behavior; the general structure of such an adaptation policy is illustrated below:

```
AdaptationPolicy id
 (Description desc)?
 (Observation id arg*)+
 (Response id arg*)+
```

Each of these policies is uniquely identified and may include a human-readable textual description. Policies also include one or more observations (also uniquely identified): when these observations are found in the knowledge-base, the entire specified collection of responses will be triggered (as in conventional knowledge-based approaches [9]). These responses are drawn from a collection of operations which modify the system's architectural model or the self-adaptive knowledge-base. Most importantly, these policy definitions are modifiable during system runtime and, given the ability to compose large-grained adaptation policies through the specification of multiple related smaller-grained ones, provide for finer control over overall system adaptive behavior.

These rule-based policy definitions are, in essence, mappings between the set of architectural observations and architectural responses. While many architectural observations will be system-specific, some observations can be expressed in semantic-free architectural terms. However, we believe that the entirety of adaptation response operations can be specified in architectural terms relating to structural and knowledge-base modifications. As illustrated in Table 1, we adopt the evolution operations outlined in [13] which we enhance with adaptation responses for the modification of observations and rules.

Overall self-adaptive behavior is governed by the collection of rule-based policy definitions and the body of knowledge collected about the system. This policy language is constrained to architectural terms: responses are limited to defined architectural and knowledge-base modifications, and triggering conditions are limited to architecturally-meaningful observations extensible by the architect for system-specific knowledge. We believe these policy definitions balance ease-of-use and simplicity with expressiveness.

Table 1. Architecture-based adaptation responses, including modification operations for architectures as well as policies.

Response	Description
AddComponent(C)	Add the indicated component, connector, or link to the architecture.
AddConnector(C)	
AddLink(L)	
RemoveComponent(C)	Remove the indicated architectural element from the architecture.
RemoveConnector(C)	
RemoveLink(L)	
AddObservation(O)	Add the specified observation or adaptation policy to the knowledge base.
AddPolicy(P)	
RemoveObservation(O)	Remove the indicated observation or policy from the knowledge base.
RemovePolicy(P)	

3.2 Supporting Facilities

The kind of dynamically modifiable, rule-based approach we espouse in this research introduces a degree of unpredictability to the self-adaptive process. In contrast to explicitly specified adaptation scripts (such as those found in [8] or [14]), these policies may exhibit rule conflicts leading to detrimental adaptations, a situation compounded by rule runtime change. Therefore, we recognize the need to provide for facilities accounting for such unpredictability. In addition to integrating existing techniques for alleviating rule conflicts from the knowledge-base research community [17], we also envision additional support in terms of architectural constraint resolution and architectural configuration management.

The constraint resolution facilities we plan on investigating relate to specific restrictions on the presence of architectural elements and their compositions. These constraints include stylistic invariants applied to entire architectural topologies (for which there exist established techniques, such as [11]) in addition to specific permissions on which architectural elements and interconnections may be modified by the adaptation knowledge-base. Architectural configuration management facilities provide support for architect-driven recovery operations from configurations arrived at through the self-adaptive process. While not directly related to our main research focus of adaptation policy specification, an investigation of these facilities is necessitated by the unpredictability introduced by our chosen methods.

4. EVALUATION METHODOLOGY

Though the supporting infrastructure for this work is still only in prototype status, we plan on refining our tools and evaluating our overall approach across a number of aspects and domains. First, we plan on investigating the *expressiveness* and *complexity* inherent in the knowledge-based approach we describe. Specifying adaptation policy as a collection of independent rules is certainly harder to conceptualize than sequential modification scripts, so we are interested in examining the degree of complexity inherent in rule-based policy specifications. Second, we intend to investigate related dimensions of *accuracy* of adaptations and the *scalability* and *overhead* imposed by the supporting infrastructure. For all self-adaptive systems, and especially for those relating to safety-critical tasks, it is important to assure that adaptations actually do take place when and as they are needed. The scalability characteristics of self-adaptive applications—as both architectural topologies and the set of adaptation policies grow—and the computational overhead of the supporting tool infrastructure are critical factors in whether this approach is realistic in a practical setting, especially in resource constrained domains.

We intend to examine these properties of our approach through activities combining the application of our methodology to both experimental as well as real-world systems. Initially, we plan on evaluating our methods on small-scale self-adaptive systems with relatively few adaptation policies, which will give us a basis for rapid experimentation and exploration of the issues involved in applying our approach. We are considering using Robocode [2], an infrastructure for the simulation and visualization of battles between robots, as the basis for this activity by designing self-adaptive systems which modify their battle strategy based on performance and the competitive field.

Finally, we plan on transitioning our methodology to more realistic applications centered on the use of self-adaptive systems in the satellite ground system domain. Through past work, we have gained insight into such systems and can draw on both personal experience as well as active industry collaborations for the application of our self-adaptive methodology to an adaptive telemetry processing ground system based on STARS [1]. The scale of such a system and the large number of adaptation policies needed to provide for needed self-adaptive capabilities will provide for a realistic evaluation of the performance characteristics of our approach.

5. CONCLUSION

One of the most important elements of architecture-based self-adaptive software is the manner in which adaptation policies are specified. Through an examination of other research approaches, we identify the need for an approach which supports policies which are strongly decoupled from systems and dynamically modifiable at system runtime.

We believe that the knowledge-based specification approach we outline in this paper satisfies these requirements. By explicitly representing adaptation policies at the architectural level, decoupling these policy definitions from both system components and architectural configurations, and supporting the runtime dynamic modification of rule-based policy specifications, we offer an approach which supports the potential for policy reuse, system flexibility, and adaptation policy dynamism. Additionally, we leverage well-developed knowledge-base techniques and expert system tools from the artificial intelligence community and contribute a novel integration of these technologies in the dynamic architecture management domain.

6. REFERENCES

[1] *Spacelift Telemetry Acquisition and Reporting System*. <http://www.aero.org/controlsystems/stars.html>, The Aerospace Corporation.

[2] *Robocode*. <http://robocode.alphaworks.ibm.com>, IBM.

[3]Allen, R.J., Douence, R., and Garlan, D. Specifying and Analyzing Dynamic Software Architectures. In *Proceedings of the 1998 Conference on Fundamental Approaches to Software Engineering*. Lisbon, Portugal, March 1998, 1998.

[4]Andrade, L. and Fiadeiro, J.L. An architectural approach to auto-adaptive systems. In *Proceedings of the 22nd International Conference on Distributed Computing Systems Workshops*. p. 439-444, 2002.

[5]Dashofy, E., Hoek, A.v.d., and Taylor, R.N. Towards Architecture-Based Self-Healing Systems. In *Proceedings of the First ACM SIGSOFT Workshop on Self-Healing Systems*. p. 21-26, ACM. Charleston, South Carolina, November 18-19, 2002.

[6]Dowling, J. and Cahill, V. Dynamic Software Evolution and the K-Component Model. In *Proceedings of the Workshop on Software Evolution, OOPSLA 2001*. 2001.

[7]Endler, M. A Language for Implementing Generic Dynamic Reconfigurations of Distributed Programs. In *Proceedings of the 12th Brazilian Symposium on Computer Networks*. p. 175-187, 1994.

[8]Garlan, D. and Schmerl, B. Model-based adaptation for self-healing systems. In *Proceedings of the First ACM SIGSOFT Workshop on Self-Healing Systems*. November, 2002.

[9]Hayes-Roth, F. The Knowledge Based Expert System: A Tutorial. *IEEE Computer*. 17(9), p. 11-28, 1984.

[10]Magee, J. and Kramer, J. Dynamic Structure in Software Architectures. *Software Engineering Notes*. 21(6), p. 3-14, 1996.

[11]Magee, J. and Kramer, J. Self organising software architectures. In *Proceedings of the Second International Software Architecture Workshop (ISAW-2) and International Workshop on Multiple Perspectives in Software Development (Viewpoints '96) on SIGSOFT '96 Workshops*. p. 35-38, 1996.

[12]Oreizy, P., Gorlick, M.M., Taylor, R.N., Heimbigner, D., Johnson, G., Medvidovic, N., Quilici, A., Rosenblum, D.S., and Wolf, A.L. An Architecture-based Approach to Self-Adaptive Software. *IEEE Intelligent Systems*. 14(3), p. 54-62, May-June, 1999.

[13]Oreizy, P. *Open Architecture Software: A Flexible Approach to Decentralized Software Evolution*. Ph.D. Thesis. Information and Computer Science, University of California, Irvine, 2000.

[14]Sztipanovits, J., Karsai, G., and Bapty, T. Self-Adaptive Software for Signal Processing. *Communications of the ACM*. 41(5), p. 66-73, 1998.

[15]Wermelinger, M. Towards a Chemical Model for Software Architecture Reconfiguration. In *Proceedings of the Fourth International Conference on Configurable Distributed Systems*. p. 111-118, Annapolis, Maryland, May 4-6, 1998.

[16]Wermelinger, M. and Fiadeiro, J.L. A Graph Transformation Approach To Software Architecture Reconfiguration. *Science of Computer Programming*. 44(2), p. 135-155, 2002.

[17]Wu, P. and Su, S.Y.W. Rule Validation Based on Logical Deduction. In *Proceedings of the Second International Conference on Information and Knowledge Management*. p. 164-173, Washington, D.C., United States, 1993.