

Issues in Modeling and Analyzing Dynamic Software Architectures

Peyman Oreizy

Information and Computer Science
University of California, Irvine
Irvine, CA 92697-3425 USA
+1(714) 824 8438
peyman@ics.uci.edu

ABSTRACT

We summarize and relate current work in the area of dynamic software architecture. Based on this and our experience to date in constructing systems based on dynamic software architectures, we present a set of open research issues that need further investigation.

Keywords

Dynamic software architectures, runtime software evolution, runtime software reconfiguration, software consistency, software integrity.

1 INTRODUCTION

Society's increasing dependence on software-intensive systems is driving the need for dependable, robust, continuously available systems. The ability to evolve a system at runtime is one critical aspect of achieving continuously availability. Although operating systems and programming languages have provided programmers with the ability to carry out runtime software changes since the 1960's, such mechanisms do not guarantee that a change will have the desired effect or maintain application integrity. It is therefore imperative that we develop approaches to runtime system evolution that help us (a) determine what to change, (b) facilitate reasoning about the consequences of a change, and (c) govern change to preserve application integrity. Without this, the risks introduced by runtime software evolution may outweigh those associated with shutting down and restarting the system for reconfiguration.

Several subdisciplines of computer science are actively investigating and promoting a wide variety of approaches to runtime software evolution. These include approaches based on programming languages (e.g., [GJB96] and [PHL97]), data-flow architectures (e.g., [GR91]), distributed systems (e.g., [KM85]), distributed object systems (e.g., dynamic object binding services in CORBA [OMG96] and COM [Broc94]), compilers (e.g., [Fra97]), operating systems (e.g., [CC98]) and real-time systems (e.g., [SRG96]).

Recently, the software architecture community has begun to investigate opportunities for utilizing architecture-based models as a basis for runtime evolution (e.g., [ADG98, KM98, MK96, OMT98,

Ore96, Wer98]). The term *dynamic architectures* denotes that the application's architecture evolves during runtime. Whereas a majority of the effort in other subdisciplines has been directed at providing appropriate mechanisms for implementing runtime change, the software architecture community has focused on modeling and analyzing runtime change in order to ensure application integrity.

The rest of this paper summarizes and relates current work in modeling and analyzing dynamic software architectures, and concludes by presenting a set of open research problems based on our experience to date.

2 CURRENT WORK IN DYNAMIC SOFTWARE ARCHITECTURES

Although other subdisciplines of computer science, and especially software engineering (e.g., analysis and testing and architecture-based analysis), have much to offer in advancing dynamic software architecture research, we report on ongoing work that has directly focused on dynamic software architectures.

We partition current work into three categories: languages and tools for modeling dynamic architectures in order to prove system properties, languages for specifying architectural changes to a running system, and tools for supporting the implementation of dynamic software architectures. Each of these is addressed in the following subsections.

2.1 Modeling Dynamic Architectures

Architecture description languages (ADLs) provide a formal basis for describing software architectures by specifying the syntax and semantics for modeling components, connectors, and configurations. Since a majority of existing ADLs have focused on design issues, they have primarily been used for static analysis and system generation. Most ADLs assume a static description of a system's architecture, and provide no facilities for specifying dynamic architectures.

Recently, several dynamic architecture description languages (DADLs) have been developed. These include a Chemical Abstract Machine-based formalism [Wer98], Darwin [MK96], Dynamic ACME [MGW97], Dynamic Wright [ADG98], and Rapide [LV95].

Common to all of these formalisms is the ability to express structural changes to the architectural model. For example, Darwin's *dyn* construct may be used to dynamically instantiate and connect new components to an architecture. These structural models are typically annotated with semantic information (e.g., component dependence or type information) to support analysis.

But purely structural models capture a very limited model of the application, and generally fail to help us ensure application integrity during the course of a runtime change. Consequentially, all of these DADLs (except Dynamic ACME) have incorporated some behavioral modeling capabilities. Dynamic Wright [ADG98], for instance, models the communication protocols between components and thus may be used to determine protocol conformance during the course of a runtime change. Darwin [MK96] has recently incorporated a behavioral model based on finite state machines that supports the analysis of liveness and safety properties during the course of a runtime change [KM98]. Wermelinger's Chemical Abstract Machine-based formalism [Wer98] elegantly unifies an application's behavioral model and the rules governing runtime changes, but its underlying formalism of rule rewriting is far removed from the conceptual model used by system designers.

Rapide [LV95] is unique in comparison with other DADLs in that it uses operational descriptions of architectures and dynamic change, whereas other DADLs use declarative descriptions. Consequentially, static analyses of Rapide descriptions are exceedingly difficult, but are ideal for simulation and collection of execution traces that may be subsequently analysis.

2.2 Architecture Modification Languages

While DADLs focus on describing software architectures for the purposes of analysis and system generation, architecture modification languages (AMLs) focus on describing changes to architecture descriptions. Such languages are useful for introducing unplanned changes to deployed systems by changing their architectural models. Examples include ArchStudio's Extension Wizard modification scripts [OMT98], C2's AML [Med96], and Clipper [AHP94]. All of these languages are operational and use similar constructs.

2.3 Runtime Support for Dynamic Architecture

Several publicly available research tools have been implemented that facilitate the construction of applications built with dynamic software architectures. These include the ArchStudio tool suite¹ from UC Irvine and the Software Architect's Assistant² from Imperial College.

ArchStudio is a tool suite that supports the interactive, graphical specification and runtime modification of software architectures described in the C2-style. The

1. <http://www.ics.uci.edu/pub/arch/dynamic-arch.shtml>

2. <http://www-dse.doc.ic.ac.uk/~kn/saa.html>

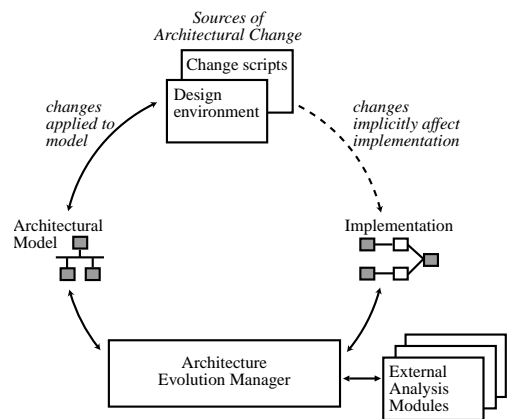


Figure 1. ArchStudio's conceptual architecture.

approach used by the ArchStudio tool suite is depicted in figure 1. Runtime changes are described in terms of the application's *architectural model* by a set of tools, such as *change scripts* and an interactive graphical *design environment*. Changes may include the addition, removal, or replacement of components and connectors, or changes to the topology of the architecture. The *Architecture Evolution Manager (AEM)* is notified of these changes and has the opportunity to revoke changes that violate system integrity, which may be determined with the aid of *external analysis modules*. If a change does not violate system integrity, the AEM makes the corresponding change to the system's *implementation*. The AEM currently uses Armani, an architectural constraint language extension to ACME provided by Bob Monroe at CMU, for preventing runtime changes that violate constraints specified on the application's architectural model.

The Software Architect's Assistant (SAA) [NK95] is an interactive graphical tool for specifying, analyzing, and constructing dynamic architectures. Architects use the SAA to graphically specify Darwin architectural models, analyze them using external verification tools, and generate skeleton code for the Regis distributed programming environment. Although SAA provides intelligent graphical layout and navigation of design elements, it does not support runtime architecture monitoring or manipulation (an earlier system called ConicDraw provided such functionality).

3 OPEN ISSUES

Our experience to date in implementing dynamic architecture-based systems, and in developing and using the ArchStudio tool suite reveals several important issues that need further investigation. These are summarized below:

- *How are runtime changes to the architectural model governed?* Behavior-based formalisms such as Darwin and Dynamic Wright do an adequate job of modeling and verifying particular functional properties of a system (such as deadlock and liveness properties, message loss, protocol conformance,

etc.), but neglect non-functional properties that may be equally important in maintaining system integrity. Non-functional constraints (such as interoperability, performance and responsiveness, and security) are more appropriately modeled using structure-based formalisms (such as Armani). As a result, approaches that integrate multiple formalisms should be pursued to ensure broad coverage of system concerns.

- *Are the model and implementation consistent with one another?* This is a concern of all model-based analysis techniques, but one that has particularly important for dynamic architectures since runtime changes to the system are described in terms of its architectural model. For example, a modeling language may assume that runtime changes are instantaneous, when in fact the implementation does not support instantaneous changes. Whether or not particular differences between the model and implementation are significant depends on the type of analysis being performed. In either case, the correspondence between the two must be ensured.
- *When can runtime changes be applied to the system?* In non-dynamic architecture-based systems, the system starts in a known initial state, which components implicitly make assumptions about during initialization. In a dynamic architecture-based system, components being introduced into the running system cannot necessarily make assumptions regarding the current state of the system. As a result, a component may have to undergo a “synchronization” period during which its internal state is made consistent with that of the rest of the system. In the case of component replacement, the incoming component could inherit the state of the outgoing component and continue processing where the outgoing component left off. The different DADLs make different assumptions about when runtime changes may be applied. For example, Darwin requires components affected by a change to be in a “quiescent” state before runtime reconfiguration occurs. Dynamic Wright, on the other hand, requires that the architect explicitly model opportunities for runtime reconfiguration by specifying special “control” events in the architectural model. Other strategies are also possible, such as requiring architect-specified constraints based on the internal state of select components to be satisfied before reconfiguration is permitted.
- *How are changes to the architectural model mapped to corresponding changes in the implementation?* Most tools currently assume a one-to-one mapping between model entities and implementation entities (e.g., each component maps to a Java class). But should other mappings, such as one-to-many, many-to-one, etc., be supported as well? How would such mappings impact dynamic architecture analysis?
- *What is the impact of architectural styles on analyzing and implementing runtime changes?* Our early experience indicates that some aspects of a style (such as the use of implicit invocation and asynchronous message passing) facilitate runtime change by reducing component dependencies and localizing the impact of runtime changes [OT98]. A broad study that investigates numerous architectural styles would better our understanding of their impact on runtime change.
- *What is the impact of architectural connectors on analyzing and implementing runtime changes?* Early evidence indicates that connectors facilitate both the analysis and implementation of runtime evolvable systems by insulating components from the effects of runtime changes [ADG98, KM98, OMT98].
- *How should non-instantaneous changes be addressed?* All of the existing DADLs except Darwin assume that runtime changes occur instantaneously — i.e., that the system moves between two “valid” configurations without the passage of time. In practice, this is rarely the case, since a complex runtime change may require the simultaneous upgrade of several interdependent components, each of which may be executing on different hosts in a network. As a result, a system’s architecture may “move” through several invalid configurations before reaching a final valid state. Although the architectural model may legitimately disallow certain configurations, doing so solely based on intermediate invalid configurations will prevent some classes of valid runtime changes. As a result, a mechanism that supports transactional changes should be provided. In Darwin, a reconfiguration manager orders components directly affected by a change *and* components directly adjacent to them to enter into a “quiescent” state. This ensures that the components directly affected by a change will not receive service requests, thereby insulating them during the course of a complex change [KM98].
- *What is the impact of supporting decentralized software evolution?* Decentralized software evolution [Ore98] introduces additional concerns since multiple, independent software vendors can make changes to the system. As a result, (a) runtime changes must be robust to variations in the running system’s architecture, (b) some consistency analysis must be performed in-the-field, and (c) the implementation must support the incorporation of external components and connectors, as well as changes to the set of rules that govern system integrity.

4 REFERENCES

- [AHP94] B. Agnew, C. R. Hofmeister, J. Purtilo. Planning for change: A reconfiguration language for distributed systems. *Proceedings of the Second Conference on Configurable Distributed Systems (CDS 2)*, 1994.
- [ADG98] R. J. Allen, R. Douence, D. Garlan. Specifying and Analyzing Dynamic Software Architectures. *Proceedings of the 1998 Conference on Fundamental Approaches to Software Engineering (FASE '98)*. March 1998.
- [Broc94] K. Brockschmidt. *Inside OLE 2*. Microsoft Press, 1994.
- [CC98] M. Clarke, G. Coulson. An Architecture for Dynamically Extensible Operating Systems. *Proceedings of the Fourth International Conference on Configurable Distributed Systems (ICCDs 4)*. IEEE Computer Society Press. May 1998.
- [Fra97] M. Franz. Dynamic linking of software components. *IEEE Computer*, vol 30, no 3, pp 74-81, March 1997.
- [GJB96] D. Gupta, P. Jalote, G. Barua. A formal framework for on-line software version change. *IEEE Transactions on Software Engineering*, vol 22, no 2, February 1996.
- [GR91] M. M. Gorlick, R. R. Razouk. Using weaves for software construction and analysis. *Proceedings of the 13th International Conference on Software Engineering*. IEEE Computer Society Press, May 1991.
- [KM85] J. Kramer, J. Magee. Dynamic Configuration for Distributed Systems. *IEEE Transactions on Software Engineering*, vol 11, no 4, pp 424-436, April 1985.
- [KM98] J. Kramer, J. Magee. Analysing Dynamic Change in Software Architectures: A Case Study. *Proceedings of the Fourth International Conference on Configurable Distributed Systems (ICCDs 4)*. IEEE Computer Society Press. May 1998.
- [LV95] D. Luckham, J. Vera. An event-based architectural definition language. *IEEE Transactions on Software Engineering*, pp 717-734, September 1995.
- [Med96] N. Medvidovic. ADLs and dynamic architecture changes. *Second International Software Architecture Workshop (ISAW-2)*, San Francisco, October 1996.
- [MGW97] R. T. Monroe, D. Garlan, D. Wile. Acme v3.0.1 Language Reference Manual. http://www.cs.cmu.edu/afs/cs/project/able/www/acme-web/v3.0/AcmeLanguageRefManual_3.0.1.html
- [MK96] J. Magee, J. Kramer. Dynamic structure in software architectures. *Fourth SIGSOFT Symposium on the Foundations of Software Engineering*. San Francisco, October 1996.
- [NK95] K. Ng, J. Kramer. Automated Support for Distributed Software Design. *Proceedings of 7th International Workshop on Computer-aided Software Engineering (CASE 95)*, Toronto, Canada, July 1995.
- [OMG96] Object Management Group. *The Common Object Request Broker: Architecture and Specification, Revision 2.0*. July 1996. <http://www.omg.org/corba/corbiiop.htm>
- [OMT98] P. Oreizy, N. Medvidovic, R. N. Taylor. Architecture-Based Runtime Software Evolution. *Proceedings of the International Conference on Software Engineering 1998 (ICSE'98)*. Kyoto, Japan, April 19-25, 1998.
- [Ore96] P. Oreizy. Issues in the Runtime Modification of Software Architectures. *Technical Report UCI-ICS-96-35*, Department of Information and Computer Science, University of California, Irvine, August 1996.
- [Ore98] P. Oreizy. Decentralized Software Evolution. *Proceedings of the International Conference on the Principles of Software Evolution (IWPSE 1)*. Kyoto, Japan. April 20-21, 1998.
- [OT98] P. Oreizy, R. N. Taylor. On the Role of Software Architectures in Runtime System Reconfiguration. *Proceedings of the Fourth International Conference on Configurable Distributed Systems (ICCDs 4)*. Annapolis, Maryland, May 4-6, 1998.
- [PHL97] J. Peterson, P. Hudak, G. S. Ling. Principled dynamic code improvement. *Yale University Research Report YALEU/DCS/RR-1135*. Department of Computer Science, Yale University, July 1997.
- [SRG96] L. Sha, R. Rajkumar, M. Gagliardi. Evolving dependable real-time systems. *IEEE Aerospace Applications Conference*. New York, NY, pp 335-346, 1996.
- [Wer98] M. Wermelinger. Towards a Chemical Model for Software Architecture Reconfiguration. *Proceedings of the Fourth International Conference on Configurable Distributed Systems (ICCDs 4)*. IEEE Computer Society Press. May 1998.