Argo: A Design Environment for Evolving Software Architectures

Jason E. Robbins **David M. Hilbert** **David F. Redmiles**

Department of Information and Computer Science
University of California, Irvine
Irvine, California 92697-3425

{jrobbins,dhilbert,redmiles}@ics.uci.edu

ABSTRACT

Software architectures evolve as the result of numerous, interrelated design decisions. At any point in an architecture's evolution, current decisions can critically affect alternatives at later stages, and each decision has the potential of requiring previous decisions to be reconsidered. Analysis techniques that provide feedback only after "complete" sequences of design decisions have been made do not directly support the evolutionary nature of the architecture design process. We present a system for architectural analysis that more closely supports evolution by providing feedback as design decisions are made.

Keywords

Domain-oriented design environments, software architecture, human cognitive needs, software evolution

INTRODUCTION

Existing approaches to architectural analysis are coarse-grained and discrete. Design decisions are entered into a formal representation. That formal representation is fed as input to analysis tools which produce output regarding properties of the representation. Finally, architects interpret the output, relate it back to design decisions embodied in the representation, and prepare the design for another iteration. In sum, existing approaches require the architect to suspend the evolution of the architecture by creating a snapshot for analysis and, consequently, to suspend or delay the decision-making process by clustering modifications between evaluation opportunities. This design process is coarse-grained, operating on whole architectures as units. The cognitive process is correspondingly coarse-grained, dealing with clusters instead of individual decisions.

In contrast to this coarse-grained, discrete approach, we propose a fine-grained, concurrent approach. Namely, we describe a design environment that uses critics to perform analysis on partial architectural representations *while* architects are considering individual design decisions and modifying the architecture. Analysis is concurrent with decision-making so that architects are not forced to suspend the architecture's evolution or cluster their decisions in preparation for analysis. Feedback from critics can be used by architects while they are considering design decisions. Furthermore, feedback is directly linked to elements of the architecture thereby assisting architects in applying the feedback in revising the design. We believe this approach more directly supports the evolutionary nature of the architecture design process and the cognitive needs of software architects.

This research is supported in part by the Air Force Material Command and the Advanced Research Projects Agency under Contract Number F30602-94-C-0218, and by the National Science Foundation under Contract Number CCR-9624846. Additional support is provided by Rockwell International. The content of the information does not necessarily reflect the position or the policy of the funders and no official endorsement should be inferred.

Appeared in Proc. 19th International Conference on Software Engineering. Boston, MA. May 17-23, 1997. Pages 600-601.

CRITICS AND CONTROL MECHANISMS IN ARGO

Traditional approaches to software analysis follow the *authoritative assumption*: they support architectural evaluation by proving the presence or absence of well defined properties. This allows them to give definitive feedback to the architect, but limits their application to late in the design process after the architect has formalized substantial parts of the architecture.

Critics are active agents that support decision-making by continuously and pessimistically analyzing partial architectures. Each critic checks for the presence of certain conditions in the partial architecture. Due to their continuous and pessimistic nature, however, care must be taken to ensure that critics do not distract the architect by providing an overwhelming volume of feedback. Criticism control mechanisms are used to control the execution of critics and manage their feedback, so as to inform the architect without distracting from the design task at hand. Critics are embedded in a *design environment* where they have access to the architecture as it is being modified and to a model of the design process as it is being enacted. Figure 1 shows an overview of Argo, our design environment for evolving software architecture. Figure 2 shows a screenshot of Argo modeling an example architecture.

The critic-based approach makes what we call the *informative assumption*: architects are capable of making design decisions, and analysis is used to support architects by informing them of potential problems and pending decisions. Critics are written to pessimistically detect potential problems. They need not go so far as to prove the presence of problems; in fact, formal proofs are often not possible, or meaningful, on partial architectures.

Critics can deliver knowledge to architects about the implications of, or alternatives to, a design decision. In the vast majority of cases, critics simply advise the architect of potential errors or areas needing improvement in the architecture; only the most severe errors are prevented outright, thus allowing the architect to work through invalid intermediate states of the architecture. Architects need not know that any particular type of feedback is available or ask for it explicitly. Instead, they

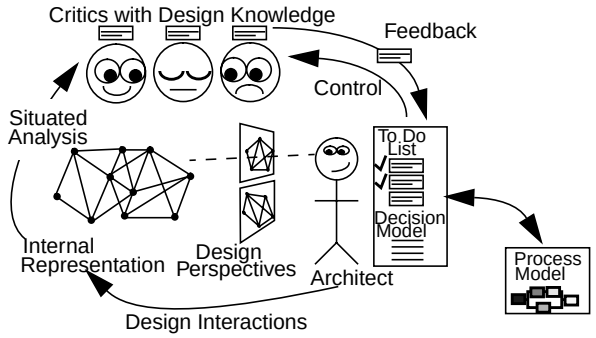

Figure 1. Design Environment Facilities of Argo

simply receive feedback as they manipulate the architecture. Feedback is often most valuable when it addresses issues that the architect had previously overlooked.

We can define a variety of potential types of critics, each type delivering a specific kind of knowledge. Correctness critics detect syntactic and semantic flaws in the partial design. Completeness critics detect when a design task has been started but not yet finished. Consistency critics detect contradictions within the design. Presentation critics detect awkward use of the notation. Alternative critics remind the designer of alternatives to a given design decision. Optimization critics suggest better values for design parameters. Some critics may be of multiple types, and new types may need to be defined, as appropriate, for a given application domain.

Criticism control mechanisms select critics for execution. During execution a critic evaluates its analysis predicate and, if appropriate, constructs a "to do" list item and posts it. Criticism control mechanisms ensure relevance and timeliness by using explicit models of the design goals and the design process. Attributes on each critic identify what type of design decision it supports. Criticism control mechanisms check those attributes against the design goals and process model. Argo's process model is an activity network, where each activity addresses design decisions of a certain type; the architect indicates which activities are currently in progress, and control mechanisms activate only timely critics.

Once critics generate design feedback, it must be presented to the architect in a usable form without distracting the architect. In Argo, the "to do" list user interface presents feedback to the architect (Figure 3). When the architect selects a pending feedback item from the upper pane, the associated (or "offending") architectural elements are highlighted in all design perspectives and details about the open design issue and possible resolutions are displayed in the lower pane.

RELATED WORK

Our focus on the cognitive needs of architects stems from the work of Fischer and colleagues [1]. In applying design environments to software architecture [4], we extended previous design environment facilities to support cognitive needs identified in the cognitive theories of reflection-in-action (via critics), opportunistic design (via a process model and "to do" list"), and comprehension and problem solving (via multiple-coordinated views [3, 6]).

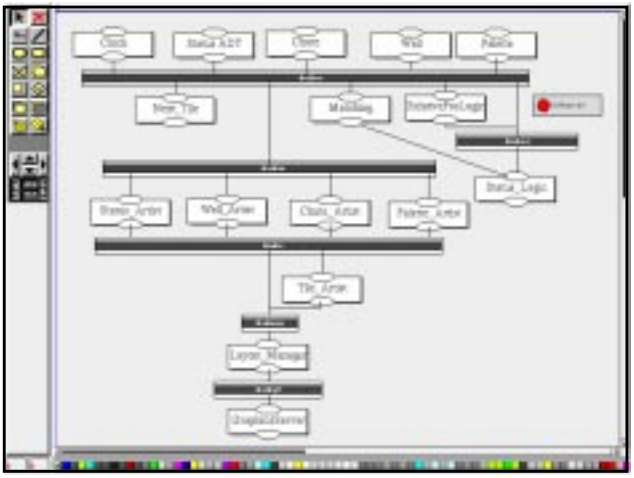
Figure 2. Conceptual Architecture Perspective

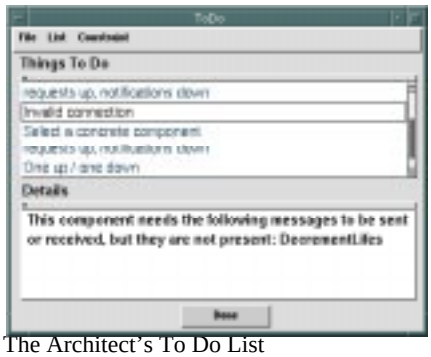

Figure 3. The Architect's To Do List

Aesop [5] is a tool that generates style-specific software architecture design environments from a set of formal style descriptions. Aesop primarily addresses requirements of architecture representation, manipulation, visualization, and analysis, without providing explicit support for evolutionary design or the architect's cognitive needs.

STATUS

It is our goal to develop and distribute a reusable design environment infrastructure that others may use, extend, and integrate with their research to better support architectural evolution and architects' cognitive needs. To date we have produced two prototypes. The initial version, coded in Smalltalk, was demonstrated at ICSE-17. The current version is implemented in Java and is available with documentation via http://www.ics.uci.edu/pub/arch.

REFERENCES

1. Fischer, G. Domain-Oriented Design Environments. *Proc. 7th Knowledge-Based Software Engineering Conference*, 204-213.
2. Fischer, G., Nakakoji, K., Ostwald, J., Stahl, G., and Sumner, T. Embedding Computer-Based Critics in the Contexts of Design. INTERCHI'93. April 1993, 157-164.
3. Kruchten, P. B. The 4+1 View Model of Architecture. *IEEE Software*. Nov. 1995, 42-50.
4. Robbins, J. E., Hilbert, D. M., Redmiles, D. F. Extending Design Environments to Software Architecture Design. KBSE'96, 63-72.
5. Shaw, M., Garlan, D. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, 1996.
6. Soni, D., Nord, R., Hofmeister, C. Software Architecture in Industrial Applications. *Inter. Conf. on Software Engineering 17*, 1995, 196-207.