To appear in the Second International Software Architecture Workshop held at SigSoft'96

Using Critics to Analyze Evolving Architectures

Jason E. Robbins David M. Hilbert David F. Redmiles
{jrobbins,dhilbert,redmiles}@ics.uci.edu
Department of Information and Computer Science
University of California, Irvine
Irvine, California 92697-3425

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. In this paper we present an approach to architectural analysis that more closely supports evolution by providing feedback as design decisions are made.

I. Introduction

Software architectures are constructed incrementally as the result of numerous interrelated design decisions made over extended periods. We visualize architecture design as a process in which a path is traced through a forking space of design alternatives. A particular software architecture can be thought of as the product of a path through this space.

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 advocate the use of 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, critic 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.

II. Overview of the Critic-Based Approach

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. Evolutionary architecture design can be better supported by the introduction of continuous, incremental analysis throughout the evolution of a complex system.

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 C2 style [16] software architecture. Figure 2 shows a screenshot of Argo modeling an example C2 style 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. If such analyses cannot be done incrementally, then they might still be packaged as critics, although their timeliness may be reduced. Alternatively, external analysis tools can be paired with "proxy critics" that remind the architect when those tools should be invoked.

The critic-based analysis approach has the potential to combine heuristic and authoritative approaches to better support evolutionary architectural design. Authoritative analysis tools provide definitive design feedback, but are often packaged as batch processes (e.g., UNIX commands) that the architect must explicitly invoke with a complete architecture as input. When heuristic critiquing is packaged as external tools (e.g., the UNIX lint tool), users often delay analysis until substantial effort has been put into formalizing tenta-

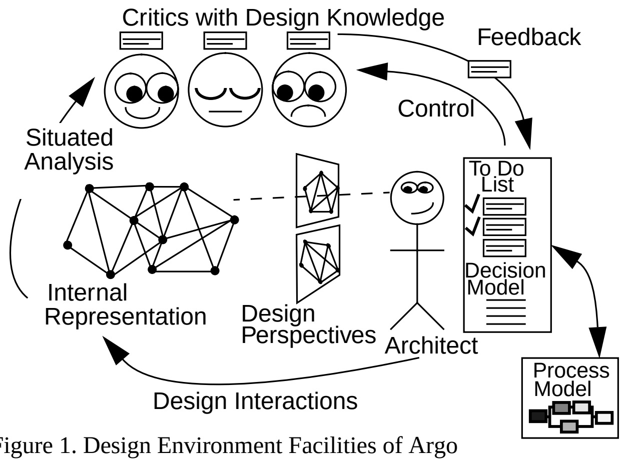


Figure 1. Design Environment Facilities of Argo

tive design decisions, and then feel overwhelmed by unmanaged feedback.

III. Characteristics of Critics

The cognitive theory of reflection-in-action [12] observes that designers of complex systems cannot conceive a design all at once. Instead, they must construct a partial design, evaluate, reflect on, and revise it, until they are ready to extend it further. Software architecture design environments can support reflection-in-action by using critics to analyze the architecture and give design feedback to the architect.

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 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. These types serve to aggregate critics so that they may be understood and controlled as groups. Some critics may be of multiple types, and new types may need to be defined, as appropriate, for a given application domain.

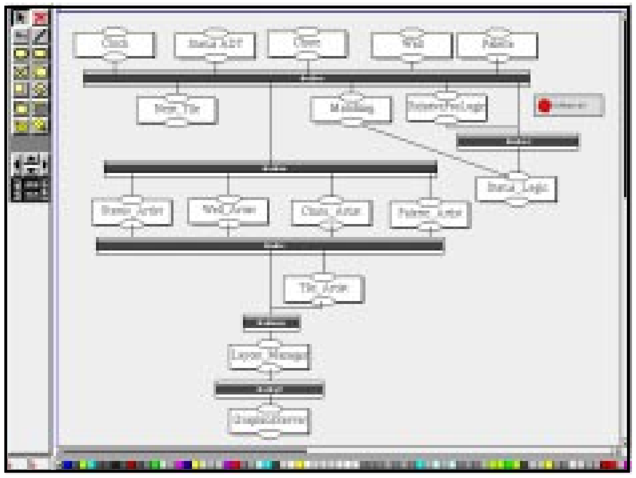


Figure 2. Conceptual Architecture Perspective

We expect critics to be invented for various reasons and by various stakeholders. Practicing architects may define critics to capture their experience in building systems and distribute those critics to other architects in their organization. Researchers may define critics to support an architectural style. Component vendors may define critics to add value to the components that they sell, and to reduce support costs. Critics may be implemented to speculate about implications of a given decision based on empirical data that indicates correlations. Also, existing literature on architectural styles and system design provides advice that can be made active via critics.

In Argo, a critic is implemented as a combination of an analysis predicate, attributes for determining relevance, and a “to do” list item to be given as design feedback. The stored “to do” list item contains a headline, a description of the issue at hand, contact information for the critic’s author, and a hyperlink to more information.

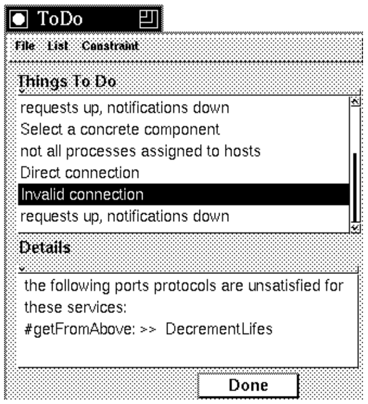
Criticism control mechanisms select critics for execution. During execution a critic evaluates its analysis predicate and, if appropriate, constructs a “to do” list item and posts it. We encode critics as programming language predicates; deciding on what languages are best for expressing critics is a topic for future research. Table 2 presents a connection checking critic in detail.

IV. Criticism Control Mechanisms

The “to do” list supports the cognitive theory of opportunistic design [9, 15, 18], which observes that designers of complex systems tend not to follow prescribed plans of action, even their own plans. The theory suggests that designers choose their next task so as to minimize the cost of contextual switching. The ordering of design tasks that minimizes costs normally cannot be known before individual design decisions are made. For example, if a decision raises design issues that require a deviation from the current process, the architect

Name of Critic	Critic Type	Decision Category	Explanation
Invalid Connection	Correctness	Connecting	Mandatory message signatures not satisfied by adjacent components in the conceptual architecture
One Up One Down	Correctness	Connecting	Violation of C2 configuration rules
Simpler Comp. Avail.	Alternative	Choosing	A “smaller” component will “fit” in place of what you have
Too Much Memory	Consistency	Resources	Calculated memory requirements exceed stated goals
Need more reuse	Consistency	Choosing	Percentage of reusable components is below stated goals
OS Incompatibility	Consistency	Choosing	Components have conflicting environmental requirements

Table 1. Selected Argo Architectural Critics



requests up, notifications down
Select a concrete component
not all processes assigned to hosts
Direct connection
Invalid connection
requests up, notifications down

the following ports protocols are unsatisfied for these services:
#getFromAbove: >> DecrementLifes

Figure 3. The Architect's To Do List

must either deviate, or mentally defer those issues in order to continue with the current process. Such process deviations may be desirable from a cognitive point of view, but they may lead designers into a variety of difficulties [6]. Fortunately, design environments can support the architect in making more informed and systematic choices as to what task ordering to follow. The "to do" list complements the process model in providing flexibility, visibility, and reminding.

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 handles 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 2). 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. Items may be filtered or sorted by various attributes, although architects may address issues in any order they choose.

V. Supporting Diverse Analysis via Critics

As the architecture evolves, new design issues will be identified, new information about the architecture will be recognized as relevant, and new critics should be written. To support evolution, the ADL should support new analyses by allowing for the addition of new attributes on individual architectural elements as needed by those analyses. In Argo, an architecture is represented as an annotated, connected graph with nodes, ports, and arcs. Several other ADLs are based on similar underlying concepts, including the C2 ADL [8] and ACME [5].

Independent critics may deliver diverse expert opinions or rules of thumb, even if they conflict. For example, one critic could advise that there are too many components at a given level of the architectural decomposition and suggest further decomposition, while another might advise that there are too many levels and suggest consolidating existing levels. Conflict per se is not a goal, but allowing conflict yields more complete support; whereas, forbidding conflict would essentially prevent architects from seeing more than one side of a design issue.

In Argo, to help organize large numbers of critics, we associate critics with the definitions of active architectural elements. Those definitions need to be loaded into the design environment only when

Attribute	Value
Name	Invalid Connection
Type	Correctness
Decision Category	Connecting
Smalltalk Predicate	`[:comp | | invalidServices | invalidServices := comp inputs , comp outputs select:[:s | s isSatisfied not]. invalidServices isEmpty not. ]`
Description	"The following port protocols are unsatisfied for these services:" <<a list of ports and services>>
More Info	http://www.ics.uci.edu/~jrobbins/
Expert	jrobbins@ics.uci.edu

Table 2. Details of the Invalid Connection Critic

those particular architectural elements are used. This limits the number of critics that are active at a given time, and allows the producers of software components to supply models of those components with embedded critics. For example, in a future software component marketplace, an architect might download several graph editing component models, try them in the current architecture, consider the resulting design feedback, and make an informed purchase decision.

A complementary approach to organizing diverse architectural knowledge is the definition of architectural styles [4, 16]. Styles define the vocabulary of the architecture and a set of rules that determine if an architecture is well formed. Architectural styles provide design guidance by suggesting constraints on design decisions. Styles may also guide analysis authors, in that critics may be grouped by style. In Argo, styles serve to organize critics, not to impose an exclusive mode on the design environment. For example, a partial architecture may nearly satisfy the rules of several styles, and the feedback from style critics related to each of those styles may be useful.

The informative assumption lowers the barrier to critic authorship so that diverse critic authors may encode their knowledge as critics. Design rationale for a given project can be made active by encoding it in critics. For example, when one architect has a negative experience with poor redraw performance using a given graph editing component, the experience can be captured in a critic that looks for the same decision in other architectures and actively provides feedback to share the experience. Under the authoritative assumption, the critic author would have to prove that the same design decision will yield poor performance in other systems; that alone may be enough to prevent the critic from being written. Furthermore, if the architect who reuses the component is not automatically presented with that feedback when it is timely and relevant, it is unlikely to improve his or her design decisions.

The effectiveness of a critic in identifying design decisions that need to be made or revised is determined by the specificity of its predicate. The effectiveness of design feedback is determined by the quality and relevance of the supplied information. The critic author is responsible for both the predicate and the information. Our approach cannot guarantee the quality of the author's work, but it does help the architect take advantage of critics from various authors.

VI. Related Work

Our focus on the cognitive needs of architects stems from the work of Fischer and colleagues [2]. In applying design environments to software architecture, 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 [7, 14]). Our extensions and their implementation are discussed in [10].

Aesop [4, 13] 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 decision-making process. For example, architectural manipulations that violate style rules may fail without providing any guidance to the architect [13]. Other software architecture design environments such as DaTE [1] and MetaH [17] also focus on systems-oriented requirements rather than the architect’s cognitive needs.

In Aesop, most analysis is performed by external tools that are explicitly invoked by the architect. Externalizing analysis eases reuse of existing analysis tools, but makes it more difficult to integrate analysis feedback into the decision-making process. Explicit invocation of analysis tools places a cognitive burden on the architect who must be aware of available analysis tools, recognize when they are appropriate, and know how their feedback relates back to the graphical depiction in Aesop. Explicit invocation of external tools scales well in terms of machine resources, but not in terms of human cognitive ability. We believe that cognitive burden alone may be enough to prevent the effective use of diverse analyses. In our terminology, Aesop currently provides no feedback management mechanisms.

VII. Summary and Future Work

One of our main contributions is that our approach is scalable in the number of critics and extensible to many new types of critics. It supports the integration of diverse analysis techniques from potentially varied sources, and addresses issues of feedback management to ensure that design feedback is timely, relevant, and manageable. The critic-based approach supports diversity by removing any assumption of cooperation among analysis developers, and by making the architect’s cognitive burden independent of the number of available analyses.

In future work we will continue the themes of our current research. Further identification of the cognitive needs of architects will lead to new design environment facilities to support those needs [11]. We intend to explore the trade-off between the depth and timeliness of feedback. In doing so we will develop a methodology for integrating external analysis tools as discussed above and investigate critics which, over the entire course of the design process, accumulate shared data and perform more in-depth analyses.

Our prototype of Argo is robust enough for experimental usage; in fact, we are using it to design the next version. However, 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 the architect’s decision-making process. Successful usage of our infrastructure by others will serve to inform and evaluate our research. 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.

VIII. Acknowledgments

The authors would like to thank Peyman Oreizy and Neno Medvidovic for their helpful comments and discussion. 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. The content of the information does not necessarily reflect the position or the policy of the funders and no official endorsement should be inferred.

IX. References

[1] Batory, D. and O’Malley, S. The design and implementation of hierarchical software systems with reusable components. *ACM Transactions on Software Engineering and Methodology*, Oct. 1992, vol.1, no. 4, 355-98.

[2] Fischer, G. Domain-Oriented Design Environments. *Proc. of The 7th Knowledge-Based Software Engineering Conference.* 204-213.

[3] 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.

[4] Garlan, D., Allen, R., and Ockerbloom, J. Exploiting style in Architectural Design Environments. *Proceedings of the Second ACM SIGSOFT Sym. on the Foundations of Software Engineering*, 1994. Software Engineering Notes, Dec. 1994, vol 19, no.5, 175-88.

[5] Garlan, D., Monroe, R., Wile, D. ACME: An Architecture Interchange Language. Technical Report CMU-CS-95-219, Dec. 1995.

[6] Guindon, R., Krasner, H., and Curtis, W. Breakdown and Processes During Early Activities of Software Design by Professionals. In: G.M. Olson ES S. Sheppard, ed. *Empirical Studies of Programmers: Second Workshop*. Norwood, NJ. 1987. 65-82.

[7] Kruchten, P. B. The 4+1 View Model of Architecture. *IEEE Software*. Nov. 1995. 42-50.

[8] Medvidovic, N., Taylor, R. N., and Whitehead, Jr., E. J. Formal Modeling of Software Architectures at Multiple levels of Abstraction. *Proceedings of the California Software Symposium (CSS’96)*, Los Angeles, CA, USA, April 1996.

[9] Rist, R. Variability in program design: the interaction of knowledge and process. *The Inter. J. of Man-Machine Studies* 1990, 1-72.

[10] Robbins, J. E., Hilbert, D. M., Redmiles, D. F., Extending Design Environments to Software Architecture Design. KBSE’96, in press.

[11] Robbins, J. E. and Redmiles D. F. Software Architecture from the Perspective of Human Cognitive Needs. *Proc. of the California Software Symposium (CSS’96)*. April 1996. 16-27.

[12] Schoen, D. *The Reflective Practitioner: How Professionals Think in Action*. New York: Basic Books, 1983.

[13] Shaw, M., Garlan, D. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, 1996.

[14] Soni, D., Nord, R., and Hofmeister C. Software Architecture in Industrial Applications. *Inter. Conf. on Software Engineering 17,* 1995, 196-207.

[15] Soloway, E., Pinto, J., Letovsky, S., Littman, D., and Lampert, R. Designing Documentation to Compensate for Delocalized Plans. *Communications of the ACM* 1988; vol. 31, no. 11, 1259-1267.

[16] Taylor, R. N., Medvidovic, N., Anderson, K., Whitehead, Jr., E. J., Robbins, J. E., Nies, K. A., Oreizy, P., and Dubrow, D. L. A Component and Message-based Architectural Style for GUI Software. *IEEE Transactions on Software Engineering*, to appear.

[17] Verstal, S. Mode changes is real-time architecture description language. In *Proc. of the Second International Workshop on Configurable Distributed Systems*, March 1994.

[18] Visser, W. More or less following a plan during design: opportunistic deviations in specification. *International Journal of Man-Machine Studies* 1990; 247-278.