# Ævol: A tool for defining and planning architecture evolution

David Garlan and Bradley Schmerl
*School of Computer Science, Carnegie Mellon University,*
*500 Forbes Ave, Pittsburgh, PA 15221.*
*{garlan,schmerl}@cs.cmu.edu*

## Abstract

*Architecture evolution is a key feature of most software systems. There are few tools that help architects plan and execute these evolutionary paths. We demonstrate a tool to enable architects to describe evolution paths, associate properties with elements of the paths, and perform tradeoff analysis over these paths.*

## 1. Introduction

Architecture evolution is a central feature of virtually all software systems. For example, many IT-based companies have evolved their systems from thin-client, mainframe-based, to three- or four-tiered architectures [3]. A similar transformation is now taking place for companies that are moving from these N-tiered systems to service-oriented architectures.

In most cases such large-scale architectural changes cannot be made overnight, and hence the architect must develop an evolution plan to change the architecture (and implementation) of a system through a series of phased releases, eventually leading to a new target system. Unfortunately, architects have few tools to help them plan and execute such evolutionary paths. While considerable research has gone into *software* maintenance and evolution, dating from the beginning of software engineering, there has been relatively little work focusing specifically on foundations and tools to support *architecture* evolution. Architecture evolution is an essential complement to software evolution because it permits planning and system restructuring at a level of abstraction where quality and business tradeoffs can be understood and analyzed.

In particular, architects have almost no assistance in reasoning about questions such as: How should we stage the evolution to achieve business goals in the presence of limited development resources? How can we reduce risk in incorporating new technologies and infrastructure required by the target architecture? How can we make principled tradeoffs between time and development effort? What kinds of changes can be made independently, and which require coordinated modifications? How can we represent and communicate an evolution plan within an organization?

Such questions require new foundations and tools that permit architects to plan, reason about, and document large-scale system-wide changes at an architectural level. Ideally these foundations would allow one to represent architecture evolution paths as first-class entities that can be expressed precisely and analyzed. They should support the expression and checking of correctness conditions (e.g., to guarantee that a proposed path satisfies certain sequencing constraints), that intermediate states of a system evolution do not introduce anomalous behavior, and that the proposed path will lead to a system with desired architectural properties. Moreover, they should allow an architect to reason not only about "correct" evolution, but also make tradeoffs to maximize business goals, such as the time to reach the target architecture and the costs involved in doing so. Finally, there should be practical tool support to automate these analyses. We are exploring these issues as part of the work described in [5].

In this demonstration we introduce a tool called Ævol that provides a platform for exploring the foundations of architecture evolution and evolution styles. This tool allows an architect to specify evolution paths, and is integrated with a software architecture design tool to allow the architectures in an evolution path to be visualized and edited. A key feature of Ævol is its support for pluggable analysis of both correctness conditions for evolution, as well as cost-benefit analysis for comparing alternative paths.

## 2. Related Work

There are four areas of related research. The first is the area of software evolution. Since the early days of software engineering there has been concern for the maintainability of software, leading to concepts such as criteria for code modularization, indications of maintainability such as coupling and cohesion, code refactoring, reverse engineering, regression testing, and many others [8]. While such advances have been criti-

*ICSE'09*, May 16–24, 2009, Vancouver, Canada
978-1-4244-3452-7/09/$25.00 © 2009 IEEE

cal to the progress of software engineering, they generally do not treat large-scale reorganization based on architectural abstractions. Working primarily in the domain of code units, they do not capture the essential high-level run-time structures necessary to reason about architectures of a complex software system. We focus on the reuse of specifications and analyses for domain-specific evolution at an architectural level.

The second closely-related area is tool support for project management and planning. For example, version control systems allow different versions of artifacts to be compared and reviewed. In most of these tools, the primary managed artifact is source code, rather than architectural structures. Consequently they do not support comparison or reasoning about different versions of the architecture. More recent research has investigated architectural versioning, focusing largely on tools to support differencing and including variants in the architectural model [1][9][10][13]. In particular, such tools are silent with respect to what might constitute a correct evolution path or a path that optimizes business goals. As such they are complementary in that we could use their approach for storing versions and integrating those with software development.

Traditional project management and software development planning approaches such as COCOMO [2] provide ways to plan and analyze software development. Focusing primarily on the end state of a maintenance or development effort, they do not provide ways to directly plan and reason about sequences of developments. General practical guidelines on organizing evolution is described in [4].

The third related area is formal approaches to architecture transformation. A number of researchers have proposed formal models to capture structural and behavioral transformation, for example category theory to describe how transformations can occur in software architecture [19]. Architecture in this sense is defined by the space of all possible configurations that can result from a certain starting configuration. Grunske [7] shows how to map architectural specifications to hypergraphs and uses these to define architectural refactorings that can be applied automatically and also preserve architectural behavior. Spitznagel in [16] focuses on architectural connector transformation to augment communication paths between components.

Recently Tamzalit and others have begun to investigate recurring patterns of architecture evolution, primarily with respect to component-based architectures [17]. They characterize patterns for updating a component-based architecture. They provide a formal approach based on a three tiered conceptual framework. Like our work, they attempt to capture recurring and reusable patterns of architecture evolution. However, unlike our work, they do not explicitly characterize or reason about the space of architecture paths, or reason about how to select appropriate paths.

The fourth related area is tradeoff analysis for architectural evolution. The work of Kazman et al. [11] applies existing architectural analysis and trade-off techniques to improve architectures. The improvements are incremental, taking into consideration only known attributes. The approach has not been considered for architecture evolution. The work in [14] proposes to use option-based techniques from economic option theory to characterize uncertainty and options available in evolution, and identifies several techniques that can then be used to calculate the points in time where introducing changes would be cost-effective in a business sense, but there is currently no tool support for it.

One important subset of work does focus on architectural evolution for specific classes of systems. Typically this work addresses architecture evolution in the context of a specific style, such as Darwin [12] and C2 [18]. Like the work proposed here, these approaches can take advantage of domain-specific classes of systems, and thereby achieve analytic leverage, as well as tool support for evolution. However, these approaches are limited to systems constructed in the particular architectural style that they support.

## 3. Ævol

What is required is a tool and approach that allows architects to plan and compare potential paths of architectural evolution. Key to the success of this approach is that it should allow the exploitation of common evolution constraints and analyses. In developing such a tool, we need to consider the following key degrees of variability in the tool:

- *Permit different analytical methods on the evolution*. There are a variety of analytic methods that can be used, from simple cost-benefit analyses to more sophisticated economics-based analyses. Different methods will require different kinds of information about the evolution to compute overall utility.
- *Exploit the domain*. It must be possible to tailor the tool to different domains and types of evolution. For example, the tool should be tailorable for evolutions that involve moving software from one datacenter to another, or for rearchitecting the system from a N-tiered style to a SOA style, etc.

We have developed a tool that functions as a platform for exploring architecture evolution. The tool, called Ævol, is a plug-in framework that supports the use of different forms of analysis and planning to be implemented and tested within the environment. Archi-

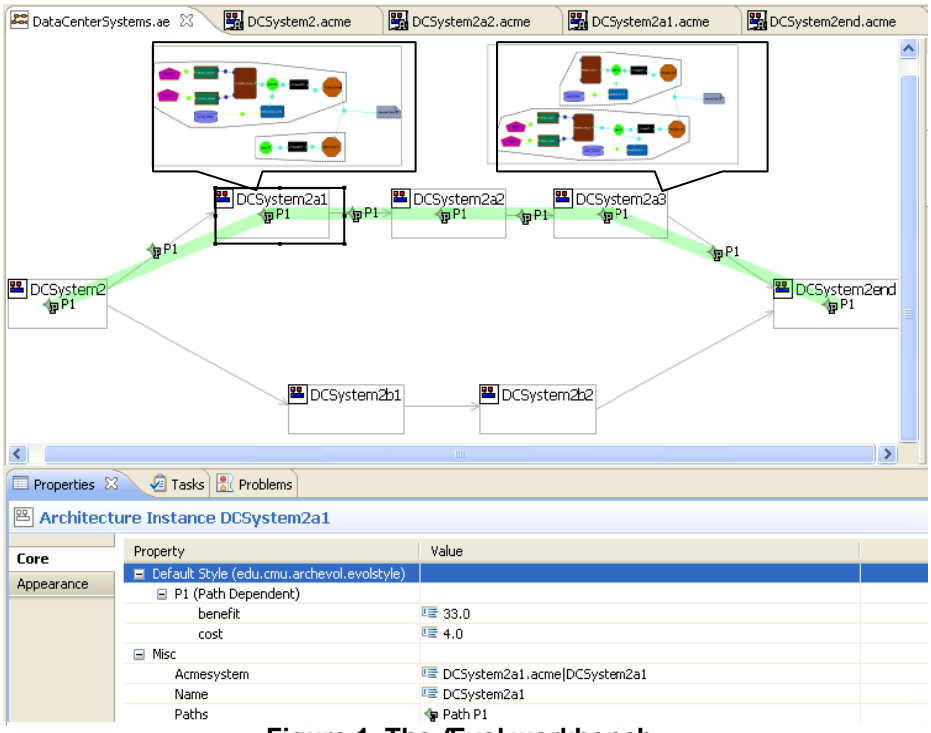


**Figure 1. The Ævol workbench.**

tects define the evolution graph in Ævol and link nodes to architectural instances that are developed in AcmeStudio [15], an editor for the Acme ADL[6]. They can then define paths and run analyses on this graph to aid in planning and choosing the optimal evolution path to take. Specifically, Ævol provides the following:

*Defining evolution graphs and linking to architectures*. Evolution graphs are directed and acyclic. Each node in the graph corresponds to an architectural instance, and each transition to an evolution step to transform the architecture at the start of the transition into the architecture at the end of the transition. The graph can have multiple termination points, and can be elaborated from the beginning, the ends, or from the middle.

*Defining evolution paths* as connected sequences of instances and transitions within an evolution graph, beginning at the start node and finishing at one of the endpoints.

*Assigning properties to instances, transitions, and paths* that enable analysis, comparison, and planning of different paths. Ævol has a rudimentary concept of *architecture evolution style*, which defines (a) constraints on evolution paths to check well-formedness; (b) analyses that can be performed over the graphs; and (c) properties that must be defined for transitions and nodes to facilitate the analysis.

*Comparing different architectural instances and their properties*. This allows architects to drill down into the differences of not only the architectural structure between steps in the evolution, but the differences in the properties of both the architectural instances and the evolution path nodes.

In addition to the base functionality outlined above, Ævol defines plug-in interfaces to allow the definition of evolution styles, different kinds of analyses, and different architecture comparison engines.

Figure 1 shows the Ævol workbench, illustrating an evolution graph. Nodes are linked to architectural instances, which can be opened in AcmeStudio. Associated with each instance is a set of properties. The selected node in the graph has the properties displayed in the Properties view at the bottom of the figure. This view displays the instances that the node is linked to, in addition to properties required for analysis (in the example in the figure, simply *cost* and *benefit*). The semi-transparent thick line in the diagram represents an evolution path. Once the properties on each of the path are filled in, it is possible to run the analysis to compute overall utility of a path and then to compare utilities of different paths.

Figure 2 shows the results of running a simple cost/benefit analysis defined by a plug-in. The calculations for each path defined in the evolution graph are listed for easy comparison. Furthermore, the results can be exported to an Excel spreadsheet, enabling further comparison. Note that the example provided here is extremely simple – the analysis in this case is based on costs and benefits of nodes, and costs of transitions.The plug-in approach, however, allows more-sophisticated analyses to be applied (for example, di-

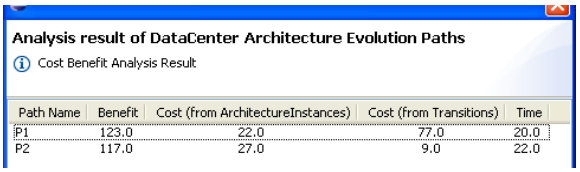


**Figure 2. Results of running a simple cost/benefit analysis plugin.**

viding costs and benefits into finer-grained elements of concern for the particular domain or business environment, accessing the results of architectural analysis to compare improvements in performance and security, defining uncertainty with each property to reflect the increased uncertainty as an architect projects the evolution further out in time, etc.)

## 4. Implementation

Ævol is written in Java as a plugin to the Eclipse framework using Eclipse's Graphical Modeling Framework. It it also a plugin to AcmeStudio architecture development environment (itself an Eclipse plug-in) to link evolution path nodes with architectural instances for each step in the evolution. Analyses are written as Java plugins using APIs provided by Ævol.

## 5. Conclusion

We demonstrate a tool for architecture evolution planning and analysis that allows architects to plan evolutionary changes to a software system from an architectural perspective. Architects can define changes to be made in each step of an evolution, and can explore multiple such evolution paths. The tool provides a plug-in framework allowing analyses so that an architect can compare and tradeoff multiple possible evolution paths. These analyses can be tailored to particular evolution domains (such as transitioning from a N-tiered architecture style to a service oriented architectural style) and to particular business environments of concern to the architect.

The plug-in approach to Ævol provides a platform on which to explore bigger evolution questions, such as how to deal with uncertainty about the future, how to better capture evolution domain knowledge, and how to guide the user to the right evolution paths. These are areas of future research.

## Acknowledgements

This work has been funded by NSF grants CNS-0615305 and IIS0534656. We gratefully acknowledge the assistance of Snehal Fulzele, Smita Ramteke, Ken Tamagawa, and Sahawut Wesaractchakit in developing Ævol as part of their Masters in Software Engineering Studio Project.

## References

[1] M. Abi-Antoun, J. Aldrich, N. Nahas, B. Schmerl, D. Garlan. Differencing and Merging Architectural Views. Automated Software Engineering Journal, **15**(1), 2008.
[2] B. Boehm. Software Engineering Economics. Englewood Cliffs, NJ. Prentice-Hall, 1981.
[3] B. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal. Pattern-Oriented Software Architecture – A System of Patterns, Volume I. Wiley, 1996.
[4] M. Erder, P. Pureur. Transitional Architectures for Enterprise Evolution. IT Professional, **8**(3):10-17, 2006.
[5] Garlan, D. Evolution Styles: Formal foundations and tool support for software architecture evolution. School of Computer Science, Carnegie Mellon TR CMU-CS-08-142, 2008.
[6] D. Garlan, R. Monroe, D. Wile. Acme: Architectural Description of Component-Based Systems. In *Foundations of Component-Based Systems*, Cambridge Univ. Press, 2000.
[7] L. Grunske. Formalizing Architectural Refactorings as Graph Transformation Systems. Proc. the 6th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing and 1st ACIS International Conference on Self-Assembling Wireless Networks (SNPD/SAWN'05). Towson, MD, 2005.
[8] C. Ghezzi, M. Jazayeri, D. Mandrioli. Fundamentals of Software Engineering. Prentice Hall 1991.
[9] A. van der Hoek, D.M. Heimbigner, A.L. Wolf. Versioned Software Architecture. In Proc. the Third International Software Architecture Workshop, pp. 73-76, Nov. 1998.
[10] A. van der Hoek, M. Mikic-Rakic, R. Roshandel, N. Medvidovic. Taming Architectural Evolution. Proc. 6th ESEC/9th ACM SIGSOFT FSE, 2001.
[11] R. Kazman, L. Bass, M. Klein. The essential components of software architecture design and analysis. The Journal of Systems and Software **79**, pp. 1207-1216, 2006.
[12] J. Magee, N. Dulay, S. Eisenbach, J. Kramer. Specifying distributed soft-ware architectures. Proc. the 5th European Software Engineering Confe-rence (ESEC'95) 1995.
[13] E.C. Nistor, J.R. Erenkrentz, S.A. Hendrickson, A. van der Hoek. ArchEvol: Versioning Architectural-Implementation Relationships. In Proc. 12th International Workshop on Software Configuration Management, pp. 99-111, Lisbon, Portugal, 2005.
[14] I. Ozkaya, R. Kazman, M. Klein. Quality-Attribute-Based Economic Valuation of Architectural Patterns. Software Engineering Institute TR CMU/SEI-2007-TR-003, 2007.
[15] B. Schmerl, D. Garlan. AcmeStudio: Supporting Style-centered Architecture Development. ICSE 2004.
[16] B. Spitznagel. Compositional Transformation of Software Connectors. PhD Thesis, School of Computer Science, Carnegie Mellon University TR CMU-CS-04-128, 2004.
[17] D. Tamzalit, N. Sadou, M. Oussalah. Evolution problem within Compo-nent-Based Software Architecture. Proceedings of the 2006 International Conference on Software Engineering and Knowledge Engineering (SEKE'06). July 2006.
[18] R. Taylor, N. Medvidovic, K. Anderson, E. Whitehead, E. Robbins, K. Nies, P. Oriezy, D. Dubrow. A component- and message-based architectural style for GUI software. IEEE Transactions on Software Engineering **22**(6), 1996.
[19] M. Wermelinger, J.L. Fiaderob. A graph transformation approach to soft-ware architecture reconfiguration. Science of Computer Programming 44:133-155, 2002.