Evolution Styles
Foundations and Tool Support for Software Architecture Evolution

David Garlan
garlan@cs.cmu.edu

Jeffrey M. Barnes
jmbarne@cs.cmu.edu

Bradley Schmerl
schmerl@cs.cmu.edu

Orieta Celiku
orietac@cs.cmu.edu

Institute for Software Research
Carnegie Mellon University
Background

• Software architecture is effective for:
  – Designing a target system
  – Evaluating a proposed or existing system
  – Communicating with stakeholders
  – Etc.

• But all of this ignores the *evolutionary* aspect of what architects do
Architecture Evolution

• **Architecture evolution** is central to software development

  - new market opportunities
  - new technologies
  - new platforms
  - new frameworks

  require architectural change

• At present, architects have few tools to help plan and execute evolutions
Architecture Evolution Questions

• How should we stage the evolution to achieve our goals, given limited resources?
• How can we be certain that intermediate releases do not break existing functionality?
• How can we make tradeoffs between time, cost, development effort, risk, etc.?
• How can we represent and communicate an architecture evolution plan?
A Model of Architecture Evolution

Current System

Target System

Time

Evolution Styles
Evolution Paths

This allows us to model:

• How the architecture *develops* over time
• The *tradeoffs* among the different ways of getting from point A to point B
• *Stages* of development, *release points*, etc.
• *Constraints* over evolution paths
• **Key insight:** Classes of domain-specific evolution paths can be *formally specified* and *reused*

• We thus introduce the notion of an *evolution style*, analogous to a traditional architectural style

• This concept allows architects to reuse their knowledge over classes of evolutions
Evolution Styles

- Thin-client/mainframe system → tiered Web application
- J2EE Web services architecture → cloud computing
- Ad hoc peer-to-peer system → hub-and-spoke architecture with coordination middleware
Example

• A company has accumulated several different subsystems, connected ad hoc by hand-coded bridge elements, which must be independently maintained
• Want to move to an off-the-shelf integration framework
Example

ReimburseExpenses
PaySalary
PaySupplier
BillCustomers
Inventory
Personnel
SupplyOffice

Accounts

initial
target

Garlan, Barnes, Schmerl, & Celiku
Evolution Styles
Evolution Styles

• This kind of migration is fairly common
• We can capitalize on past experience using an *evolution style*, which would:
  – Identify the essential characteristics of the *initial and target architectures*
  – Include a set of architectural *operators* that may be composed into an evolution path
  – Specify a set of *path constraints* that would capture correctness conditions for a path
Evolution Styles

Formally, an evolution style comprises:

• An initial architectural style
• A target architectural style
• A set of architectural operators, which transform the structure of a system
• A set of path constraints
• A collection of analyses
Example Path Constraints

• In every release of the software, all the original functionality must exist.
• The system must start in style $S_1$, progress to a hybrid style of $S_1$ and $S_2$, and end in $S_2$.
• Once a component is in data center 2, it must remain in data center 2.
Linear Temporal Logic (LTL)

- **□p (always p)**: p holds at every subsequent step.

- **◊p (eventually p)**: p holds at some subsequent step.
  - Diagram: p

- **○p (next p)**: p holds in the next step.
  - Diagram: p p p p p

- **p U q (p until q)**: p holds until q first holds.
  - Diagram: p p p p q
LTL Example

LTL is sufficient to express many interesting properties

The billing component will not be removed until a controller is introduced

\[ \text{billingComponentPresent(system)} \lor \text{controllerPresent(system)} \]

predicates over systems, defined by the evolution style

keyword that refers to the architecture of the current state
Limitations of LTL

But some properties are impossible to express in LTL

*In every release, all original functionality must exist*

\[ \Box (release \rightarrow hasAllFunc(system, __.system)) \]

Need some way to refer back to a previous step
Our Solution: Rigid Variables

Rigid variables \([\text{Ric92}]\) preserve information from previous steps

\[
\{s\} □ (\text{release} \rightarrow \text{hasAllFunc}(\text{system}, s.\text{system}))
\]

In every release, all original functionality must exist
Evaluation Functions

• In addition to hard constraints, we have evaluation functions, which help architects choose among candidate paths.

• We associate:
  – *Benefits* with *release nodes*
    • Features, quality attributes
  – *Costs* with *transitions*
    • Time, effort, money
We have developed a tool, Ævol [Gso9], that lets architects:

- Define and analyze evolutions
- Compare nodes
- Evaluate paths
- Create styles
- Enforce constraints
Future Work

- Develop an understanding of evolution operators
- Create evolution analyses
- Enhance our tool
  - Better support for operators
  - Catalog of styles
  - Visualization improvements
- Automatically generate possible paths (as opposed to merely selecting among paths)
Related Work

• Project planning
• Architecture transformation
  – Formal methods for architecture transformation
  – Tamzalit et al., evolution styles [TOGSo6]
• Tradeoff analysis for architecture evolution
  – Kazman et al., tradeoffs with tactics [KBKO6]
• Architecture evolution for specific styles
Questions and Comments

• The slides for this talk are available at http://www.cs.cmu.edu/~jmbarnes/papers/wicsa09-slides.pdf

• We would like to thank:
  – The master’s students who worked on the Ævol tool
  – Our colleagues at the Software Engineering Institute who have worked with us on architecture evolution [see CDGG09]
  – The other members of our ABLE group, who provided helpful feedback
  – Our funders, the National Science Foundation and the U.S. Department of Defense

• References


[Ric92] Richardson, “Supporting lists in a data model,” in Proc. VLDB’92

[TOG06] Tamzalit et al., “Updating software architectures,” in Proc. SERP’06