Recommendations for Architecture-Centric Software Supporting Self-Adaptive Behavior

John Georgas
Institute for Software Research
University of California, Irvine
Presented at GSAW 2003

Outline

- **Background and Context**
- Architecture as an Evolution Blueprint
- Architectural Representation
- Component-Based Architecture
- Quick Summary
- References

Architecture-Based Self-Adaptive Software

- Software that modifies itself in real-time to meet new demands or address failures.
  - Examples:
    - Replacing a failed component with a lesser capable one to maintain nominal system behavior.
    - Adding components and connectors to a running system to meet new demands.
    - Replacing components with updated ones implementing updated capabilities.
- Architecture-Based
  - Reasoning about system and adaptation policies is done solely on the basis of the architectural description.
  - Adaptation operations are expressed in terms of the high-level architectural elements (components and connectors).

Outline

- Background and Context
- **Architecture as an Evolution Blueprint**
- Architectural Representation
- Component-Based Architecture
- Quick Summary
- References

Architecture as an Evolution Blueprint

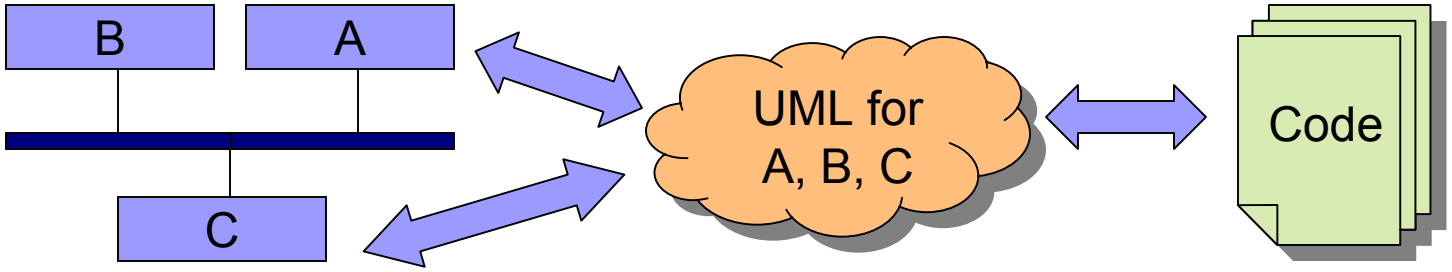
- Decisions about architectural design must precede other design concerns.
  - Component granularity, units of communication, interconnection strategies.
  - Architectural design decisions may either enable or prohibit certain implementations.
- Architecture **drives** – not only supports – the entire software lifecycle including deployment.
- Essential for this is an **architecture-to-implementation mapping**.
  - Without this, there's no point in using high-level software architectures!
    - Architecture-level analyses do not hold for the final system without a strict mapping.
    - Self-adaptive behavior cannot be architecture-based without this mapping.
    - This mapping makes architectural descriptions an actual part of the final system.
- Recommendations
  - Architecture-to-implementation *mappings* must be a part of any modeling method or language that is used to represent software architectures.
  - These mappings must be maintained and kept consistent throughout the software lifecycle.
  - Consistency must be *enforced* by tools, which somewhat increases the burden during implementation as options are limited.
  - But, some implementation artifacts will be *generated*, which somewhat lightens the load (this also helps maintain consistency).

Outline

- Background and Context
- Architecture as an Evolution Blueprint
- **Architectural Representation**
- Component-Based Architecture
- Quick Summary
- References

Architectural Representation

- Is UML an ADL?
  - An old, still discussed question.[1,2]
  - Yes, UML can be extended to describe architectural concepts.
  - But, an *ADL* is better at it.
  - So, why not use the modeling technique that is best for what you're modeling?



- Is either UML or an ADL alone sufficient? I will claim "no."
- Recommendations
  - Use a combination of an ADL and UML for software modeling.
    - An ADL for architectural concepts.
    - UML for design concepts.
  - Analyze each individually for what it's best at modeling.
  - Continuing off the previous discussion, maintain consistency of the relationship between the ADL, UML, and the implementation.
  - Wait! More abstractions, and more relationships to keep track of!
    - Yes, but your modeling capabilities are increased. The effort is worth it.

Outline

- Background and Context
- Architecture as an Evolution Blueprint
- Architectural Representation
- **Component-Based Architecture**
- Quick Summary
- References

Component-Based Architectures

- Component-based architectures with well-defined interfaces are a good start.
- What about the interactions between components?
  - Do they always take place the same way?
  - Is the unit of communication a method call, or a message?
- Recommendations
  - Connectors – and there are many “flavors” of them – must be modeled as a first-class architectural element![3]
  - Connectors should encapsulate component interaction, and reveal how different interconnection strategies with the same components can result in perhaps radically different behavior.

Outline

- Background and Context
- Architecture as an Evolution Blueprint
- Architectural Representation
- Component-Based Architecture
- **Quick Summary**
- References

Summary of Recommendations

- Architecture-to-implementation mappings are essential.
- An ADL combined with UML will produce the best modeling, and analytic results.
- Connectors must be treated as first-class entities, and be explicitly represented.

Outline

- Background and Context
- Architecture as an Evolution Blueprint
- Architectural Representation
- Component-Based Architecture
- Quick Summary
- **References**

References

1. “Is UML an Architecture Description Language” OOPSLA99
2. “Reconciling the Needs of Architectural Description with Object-Modeling Notations” Garlan, Kompanek 2000.
3. “Towards a Taxonomy of Software Connectors” Mehta, Medvidovic, Phadke.