Assessing UML's Suitability for Modeling Software Architectures

Nenad Medvidovic
Computer Science Department
University of Southern California
Los Angeles, CA 90089-0781
neno@usc.edu
http://sunset.usc.edu/~neno/

Presentation Overview

- Motivation
- Overview of UML
- Overview of the C2 Architectural Style
- Integration Strategies
- Strategy #1
- Strategy #2
- Lessons Learned
Assessing UMCs Suitability for Modeling Software Architectures

Motivation

- Community fragmentation
- Academic approach to architectures
  - focus on analytic evaluation of architectural models
  - rigorous modeling notations
  - powerful analysis techniques
  - depth over breadth
  - special-purpose solutions
- Industrial approach to architectures
  - focus on wide-range of development issues
  - families of models
  - architecture as the “big picture” in development
  - practicality over rigor
  - breadth over depth
  - general-purpose solutions

Standardization

- Provides an economy of scale
  - more and better tools
  - improved tool interoperability
  - more skilled developers
  - lower training costs

→ Combine the benefits of powerful, specialized notations with those of widely adopted, general notations

- Specific solution: integrate ADLs with UML
  - two integration approaches
  - multiple ADLs to date
Unified Modeling Language
- Large, useful set of predefined constructs
- Extensible
- Semi-formal definition of syntax and semantics
- Potential for
  - wide adoption
  - standardization
  - substantial tool support
- Basis in experience with mainstream development methods

Overview of UML
- A UML model comprises several partial models addressing
  - classes with attributes, operations, and relationships
  - states and behavior of individual classes
  - packages of classes and their dependencies
  - example scenarios of system usage
  - object instances with actual attributes in a scenario
  - actual behavior of interacting instances in a scenario
  - distributed component deployment and communication
- UML syntax and semantics are defined via
  - a meta-model
  - descriptive text
  - constraints
Extensibility in UML

- New constructs may be added to address new development issues
- Three extensibility mechanisms
  - constraints — semantic restrictions on design elements
  - tagged values — allow addition of new attributes to elements
  - stereotypes — named grouping of constraints and tagged values

  **Stereotype Person** for instances of meta-class Class

  ```
  A Person can be either female or male
  personGender : enum ( female, male )
  ```

- The metamodel may also be extended
  - results in a new notation
  - incompatible with UML-compliant tools

Overview of the C2 Architectural Style

- C2-style architectures are networks of concurrent components linked together by connectors
  - no component-to-component links
  - "one up, one down" rule for components
  - connector-to-connector links are allowed
  - "many up, many down" rule for connectors
  - message-based communication
    - requests up
    - notifications down
Integration Strategies

**Strategy #1**
- use UML "as is"
- enables direct comparison of UML and an ADL
- clarifies the relationship between architecture and design

**Strategy #2**
- use UML's built-in extension mechanisms on meta-classes
- allows automated conformance checking
  - select a meta-class semantically close to an ADL construct
  - define a stereotype and apply it to meta-class instances
  - class semantics are constrained to that of the ADL

---

Strategy #1

- Simultaneous consideration of architecture composition rules and UML notational constructs
- Develop a UML domain model
- Develop an (informal) architectural diagram
- Map domain classes to architectural components
- Design class (component) interfaces
- Provide constructs for modeling connectors
  - connectors add no functionality at the domain model level
- Model architectural structure in class diagrams
Assessing UML’s Suitability for Modeling Software Architectures

Strategy #1 Example

- **Identify** UML constructs similar to each major architectural construct
  - operation
  - message
  - event
  - port
- **Constrain** those UML constructs via stereotypes
- **Describe** semantics using UML statechart diagrams
- **Use** stereotypes to model structural aspects of an architecture
- **Use** statecharts to model semantics
Discussion

- Both integration approaches have merits and shortcomings
  - "straight" UML \(\rightarrow\) understandable architectures
  - "straight" UML \(\rightarrow\) manipulable by standard tools
  - "straight" UML \(\rightarrow\) architectural constraint violations
  - "constrained" UML \(\rightarrow\) ensures architectural constraints
  - "constrained" UML \(\rightarrow\) requires standard tool modifications
- Select the appropriate approach judiciously
- Goal
  - develop heuristics to help one determine when to use each
- ADLs and UML can complement each other
### Discussion — UML vs. ADLs

- **Similar software modeling philosophies**
  - neither constrains system implementation
  - ADL behavior modeling notations are expressible as sequence, collaboration, statechart, or activity diagrams

- **Different assumptions**
  - UML's intended usage is not architecture modeling
  - UML may make assumptions non-existent in ADLs

- **Problem domain modeling**
  - including relevant information early is critical
  - domain model as a useful architectural perspective

- **Architectural abstractions**
  - the goal is to match an architect's mental model

- **Architectural style**
  - requires constraining the UML meta-model