Modeling Software Architectures in UML

Nenad Medvidovic

Computer Science Dept.
University of Southern California
Los Angeles, CA 90089-0781

neno@usc.edu
sunset.usc.edu/~neno/
Presentation Overview

- Motivation
- Overview of UML
- Integration Strategies
  - Strategy #1
  - Strategy #2
  - Strategy #3
- Reconciling Architectural View Mismatches
- Lessons Learned
## Motivation

- Community fragmentation

<table>
<thead>
<tr>
<th>Academic approach to architectures</th>
<th>Industrial approach to architectures</th>
</tr>
</thead>
<tbody>
<tr>
<td>focus on analytic evaluation of architectural models</td>
<td>focus on wide-range of development issues</td>
</tr>
<tr>
<td>individual models</td>
<td>families of models</td>
</tr>
<tr>
<td>rigorous modeling notations</td>
<td>practicality over rigor</td>
</tr>
<tr>
<td>powerful analysis techniques</td>
<td>architecture as the “big picture” in development</td>
</tr>
<tr>
<td>depth over breadth</td>
<td>breadth over depth</td>
</tr>
<tr>
<td>special-purpose solutions</td>
<td>general-purpose solutions</td>
</tr>
</tbody>
</table>
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
  - three integration approaches
  - multiple ADLs to date
Unified Modeling Language

- Large, useful set of predefined constructs
- Extensible
- Semi-formal definition of syntax and semantics via
  - a meta-model
  - descriptive text
  - constraints
- Potential for
  - wide adoption
  - standardization
  - substantial tool support
- Basis in experience with mainstream development methods
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

[1] A Person can be either female or male

```plaintext
personGender : enum { female, male }
```

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

- **Strategy #3**
  - Extend UML to directly support architectural concerns
Strategy #1 — UML “As Is”

- 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
UML Metamodelling Architecture View

- Meta-Meta Model
- Meta Model
- Model
- User Objects
Strategy #1 Example

Modeling Software Architectures in UML

Denver, CO
April 13, 1999

Neno Medvidovic
Strategy #2 — Constrained UML

- Identify UML meta-classes semantically close similar to each major architectural construct
  - operation
  - message
  - event
  - port
  - component
  - connector
  - architecture

- Define stereotypes and apply them to meta-class instances
  - use stereotypes to model structural aspects of an architecture

- Describe semantics using UML diagrams
  - sequence
  - collaboration
  - statechart
  - activity
UML Metamodelling Architecture View

- Meta-Meta Model
- Meta Model
- Model
- User Objects
Strategy #2 Example

Attendee-1  Attendee-2  Attendee-3  Important Attendee-1  Important Attendee-2
AC

Meeting Initiator

<<Component>> Attendee-1  <<Component>> Attendee-2  <<Component>> Attendee-3  <<Component>> Important Attendee-1  <<Component>> Important Attendee-2

<<Connector>> AC

<<Connector>> MC

<<Connector>> IAC

<<Component>> Meeting Initiator
Strategy #3 — Augmented UML

- Introduce *explicit* architectural constructs and constraints in UML
- Introduce additional notations for modeling architectural semantics
- Follow an approach similar to strategy #1 to model specific architectures
- Follow an approach similar to strategy #2 to model specific architectural styles
UML Metamodelling Architecture View

- Meta-Meta Model
- Meta Model
- Model
- User Objects
Discussion

- All three approaches have merits and shortcomings
- "Straight" UML
  - understandable architectures
  - manipulable by standard tools
  - architectural constraint violations
- "Constrained" UML
  - ensures architectural constraints
  - requires complete style specifications
  - requires OCL-compliant tools
- "Extended" UML
  - provides “native” support for architectures
  - requires backward tool compatibility
  - may result in incompatible UML versions
Architectural View Mismatches

Example 1

Possible Mismatch: dependency of Flight to Flight Controller not reflected in lower level view
Architectural View Mismatches

Example 2

State Diagram for Class Screen

- User input
- Enter pressed
- Validate input
- ID invalid
- ID valid
- Check patient DB
- Create visiting record

Possible Mismatch: creation of Patient not visible in Screen

Screens

- Validate
- Get patient data
- [patient not found]
- Create
- Create
Architectural View Mismatches

Example 3

Potential Mismatch:
Link between component Account and connector transaction bus not reflected in design view

Potential Mismatch:
Link between components DB Manager and DB Admin violates C2 architectural constraints
Architectural View Mismatch Reconciliation

- Different UML diagrams present different system views
  - information represented in the views is often redundant
  → *Key challenge is to ensure inter-view consistency*

### View Analysis

**System Model**
e.g. UML model

**View Synthesis**
(graphical and textual)

**Mapping**
(Cross-Referencing)
- through names
- through patterns
- through association

**Differentiation**
(Comparison)
identify differences between model, rules, and constraints

**Transformation**
(Extraction)
- through abstraction
- through patterns
- through translation
Observations

- Software modeling philosophies
- Assumptions
- Problem domain modeling
- Architectural abstractions
- Architectural style
- Architectural views