Model-Driven Documentation of Software Architectures

Reconstructor Meeting
Delft, The Netherlands
Bas Graaf and Arie van Deursen
December 21, 2006

SWERL - Software Evolution Research Lab
http://swerl.tudelft.nl/

b.s.graaf@tudelft.nl
Introduction: MDE @ ASML

- Model-driven development for wafer-scanner control components
- Current status:
  - Modelling and documentation using UML
  - Code generation based on XMI representation using XSLT
- Problem: platform not stable yet
- Alternative: define domain-specific (MOF-based) metamodel and use model transformations

→ What about visualisations? What about documentation?
Introduction: Problem

- Upcoming of MDA: DSML vs. UML
- DSML suited for automated SE tasks:
  - Code generation
  - Analysis
  - Model transformations
- Definition of DSML abstract syntax is ‘easy’ (MOF / EMF)
- Definition of DSML notation is not so ‘easy’ (despite GEF / GMF)

- Solution: combine DSML (models) and UML (diagrams)
  → experiment in domain of software architecture
Content

• Background
  • Software architecture modelling and documentation
  • Enabling technologies
• IEEE 1471 revisited: MDAV
• Application: CaPiTaLiZe, ASML
• Pros and cons
• Discussion
SA Modelling: ADL’s

• Architecture description languages (ADL’s)
• Structure: configuration of components and connectors
• Behaviour: interacting components and connectors
• In-depth description of software architecture: formal syntax and semantics
• Analysis / code generation
• Immature: largely ignored by industry
SA Documentation: Views

- Architectural view per set of concerns:
  e.g., module-uses view, pipe-and-filter view
- Conform viewpoint
- View = diagrams (primary presentation) + text
- UML
- Wide-range of issues
- Documentation / communication / (informal) assessment
- Industry standard: IEEE 1471
MDAV: Model-Driven Architectural Views

- Architectural modelling using ADL’s
- Architectural documentation using UML Diagrams
- Mapping using model transformations

- Requires: ADL metamodel, means to create models, mapping to UML

IEEE 1471 revisited
Enabling Technologies

• MOF + EMF for metamodelling
• UML for modelling (diagrams)
• XML/XMI for model serialisation
• ATL for model transformations
Example: CaPiTaLiZe (ACME)

capitalize → CaPiTaLiZe

Diagram:
- **DataSource**
- **split**
- **upper**
- **lower**
- **merge**
- **DataSink**
CCADL → UML

Garlan et al. [2002]:
- Component → Class
- ComponentType → Stereotype
- Connector → Association
- ConnectorType → Stereotype
- Input / output Port → Interface

**Rule**: Interface

```plaintext
unique lazy rule Interface {
  from port:CCADL!Port(port.roles->isEmpty())
  to int:UML!Interface(name = if port.type.name = 'filterIn' then 'input' else 'output' endif),
  gen:UML!Generalization(parent = int, child = CCADL!Component->allInstances() -> select(c | c.ports->includes(port)))
}
```

**Unique lazy rule**: Stereotype

```plaintext
unique lazy rule Stereotype {
  from comp:CCADL!Component
to st:UML!Stereotype (name = comp.type.name, baseClass = 'Class')
}
```

**Rule**: Class

```plaintext
unique lazy rule Class {
  from comp:CCADL!Component
to class:UML!Class(name = comp.name, stereotype = thisModule.Stereotype(comp),
  ownedElement = comp.ports->select(p | p.roles->isEmpty()) -> collect(e | thisModule.resolveTemp(e,'gen')))
}
```
ASML: Supervisory Machine Control

- Code generation based on task-resource models
- Metamodel defines Task, Resource, Capability, Behaviour, etc …
- Documentation using UML Class Diagrams and UML Activity Diagrams
- TR metamodel → UML metamodel
Pros and Cons

• Benefits
  • Combine UML with DSML’s
  • Only abstract syntax is required for DSML (metamodel)
  • Take advantage of existing UML/ADL tooling and ADL2UML mappings
  • Implementation and documentation evolve simultaneously
  • Lightweight introduction of MDE (transitional phase)

• Drawbacks
  • Semantic mismatch between UML and DSML (use stereotypes)
    • Documentation requirements
    • DSML semantics
  • No graphical model editor
  • Practice: tooling incompatibilities (XMI/UML/MOF versions)
Discussion

- Bridge: grammars → metamodels

- Alternatives:
  - No documentation
  - Separately create documentation (manually)
  - Develop graphical editors
Thank You
rule Association {
    from conn:CCADL!Connector
    to asoc:UML!Association(
        name <- '<<' + conn.type.name + '>>',
        connection <- Set{conn.roles->asSequence()->first(),
        conn.roles->asSequence()->last()})
}
rule AssociationEnd {
    from r:CCADL!Role
    to aend:UML!AssociationEnd(
        isNavigable <- r.type.name='pipeOut',
        participant <- CCADL!Component->allInstances()->
        >select(c|c.ports->includes(r.port))->asSequence()->first())
}