Inconsistency Management in Model-Driven Engineering

Ragnhild Van Der Straeten (SSEL)
Inconsistency Management in Model-Driven Engineering

• Model-driven Engineering
  – Primary assets are models
  – Model: views on the software system on a certain level of abstraction

• Inconsistency Management
  – Inconsistencies
    • Different models developed by different persons
    • Interdependencies between models poorly understood
    • Requirements unclear or ambiguous
    • Models are incomplete

How to manage inconsistencies?
Overview

• THE PAST
  – Classification of inconsistencies
  – Description Logics for detection and resolution
  – Graph Transformation for interdependencies
  – Tool Support
## Classification of Inconsistencies

<table>
<thead>
<tr>
<th></th>
<th>Behavioural</th>
<th>Structural</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Specification</strong></td>
<td>Invocation interaction</td>
<td>Dangling type reference</td>
</tr>
<tr>
<td></td>
<td>Observation interaction</td>
<td>Inherited cyclic composition</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Connector specification missing</td>
</tr>
<tr>
<td><strong>Specification - Instance</strong></td>
<td>Specification incompatibility</td>
<td>Instance specification missing</td>
</tr>
<tr>
<td></td>
<td>Specification behaviour</td>
<td></td>
</tr>
<tr>
<td></td>
<td>incompatibility</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Invocation behaviour</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Observation behaviour</td>
<td></td>
</tr>
<tr>
<td><strong>Instance</strong></td>
<td>Invocation inheritance</td>
<td>Disconnected model</td>
</tr>
<tr>
<td></td>
<td>Observation inheritance</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Instance behaviour incompatibility</td>
<td></td>
</tr>
</tbody>
</table>
Description Logics

Knowledge base

Terminology (Tbox)

Father = Man \( \sqcap \exists \text{has\_child}. \top \)  
Human = Mammal \( \sqcap \) Biped

Concrete Situation (Abox)

John : Human \( \sqcap \) Father  
John has\_child Bill
UML Models as DL KBs

- Tbox representing UML metamodel
  - individual-of
  - Abox representing user-defined UML models

- class diagram
  - spanning functions for UML PSM concepts
  - call sequences
  - sequences

- PSMs
  - spanning functions for UML trace concepts

- Tbox representing PSMs
  - Tbox representing call sequences
  - stimuli, links and slots
  - individual-of

- objects, links and slots
  - individual-of

- Tbox representing classes
  - Tbox representing SD traces
UML Models as DL Knowledge Bases

Tbox representing UML metamodel

Individual of

Class diagrams
PSMs
Sequence diagrams

Aboxes representing user-defined models

Tbox representing PSM

Tbox representing classes

Tbox representing sequences
Detection of Inconsistencies

\[
\text{ans}(psm, cl, op) \\
\quad \leftarrow \text{protocolstate}(psm) \land \text{contextbehavior}(psm, cl) \land \\
\quad \text{regions}(psm, r) \land \text{transitions}(r, t) \land \text{protocoltransition}(t) \land \\
\quad \text{refereedoperation}(t, op) \land ((\neg \text{has_known_successor}(\text{operatoin}, op)) \lor \\
\quad ((\neg \text{operatoin}(cl, op)))) \land (\neg \text{general}(\text{superc, c})) \land \text{operatoin}(\text{superc, op}))
\]
Resolution of Inconsistencies

- Rule-based Inconsistency Resolution Approach

Condition:
- check inconsistency
- information about state of the model
- user interaction

Conclusion
- resolution actions: add/delete instances, fillers
Inconsistency Resolution: Iterative & incremental

- Induced Inconsistencies
  - Resolution of one inconsistency may induce other inconsistencies as a side effect

- Conflicting Resolutions
  - Some resolutions may not be jointly applicable

- Resolution Cycles
  - Certain sequences of resolutions may reintroduce a particular inconsistency occurrence
Graph Transformations: Critical Pair Analysis

• $T_1$ and $T_2$ form a critical pair if
  – they can both be applied to the same initial graph $G$
  – applying $T_1$ prohibits application of $T_2$

• $T_2$ sequentially depends on $T_1$ if
  – $T_1$ can be applied to $G$ but $T_2$ cannot
  – applying $T_1$ triggers application of $T_2$
Tool Support

• Poseidon plug-in
  – but:
    • Translation UML meta -> DL manually
    • UML 1.x

• Eclipse plug-in (was planned for 2006/Q4, 2007/Q1)
  – Usage of Ecore, EMF, UML2 editor(?), AGG
  – Automatic translation UML meta -> DL
Overview

• THE PAST
  – Classification of inconsistencies,
  – Description Logics for detection and resolution,
  – Graph Transformation for interdependencies,
  – Tool Support.

• THE FUTURE
  – General inconsistency management framework
    • Classification, comparison of formalisms,
    • Different modeling languages,
    • Bridging the gap between theory and practice.
Classification, comparison and integration of formalisms

- **Our research**
  - DL \(\leftrightarrow\) Graph Transformations

- **Goal**
  - integrate variety of formal approaches
  - classify, compare and combine different formalisms for inconsistency management

- **Collaboration**
  - partners working in the area of inconsistency management or studying certain formalisms.
  - compare, integrate formalisms.
Different Modeling Languages

- **Our research**
  - Until now: UML class, sequence and state machine diagrams

- **Goal**
  - Incorporate other modeling languages, e.g., workflow languages and their AO extensions.
  - Which kinds of inconsistencies?
  - Formalism for detection and resolution?
  - Integrate, compare, … with other formalisms.

- **Collaboration**
  - Defining semantics of these MLs.
  - Defining right formalism.
Bridging the gap between theory and practice

- **Goal**
  - Based on comparison of formalisms:
    - Improve existing approaches
  - Use, apply formal techniques of other partners from other WPs

- **Research**
  - Integrate these approaches in our framework and provide feedback to the partners developing/investigating these formalisms.

- **Collaboration**
  - Partners studying certain formalisms -> provide these formalisms for inconsistency management.