Inconsistency Management in Model-Driven Engineering

An Approach using Description Logics

Ragnhild Van Der Straeten

Advisors: Prof. Dr. Viviane Jonckers and Prof. Dr. Tom Mens
Model-Driven Engineering

Primary assets are *models*

*Model transformation* is the heart and soul of MDE
The UML in MDE

• The Unified Modeling Language
  • has become the de facto modelling language
  • captures knowledge at different abstraction levels
  • covers “any” system
  • evolved into UML 2.0
  • different diagram types:
    • class diagrams, composite structure diagram, component diagram, deployment diagram, object diagram, package diagram, activity diagram, use case diagram, state machine diagram, interaction overview diagram, sequence diagram, communication diagram, timing diagram.

• abstract syntax expressed by class diagrams.
INCONSISTENCY!!
Problem Statement

• Inconsistencies can arise due to:
  • coexistence of different diagrams that overlap,
  • different overlapping submodels,
  • refinement of models,
  • evolution of models.

• Poor support for inconsistency management in state-of-the-art UML CASE tools.

• UML is huge and lacks semantics.
## Inconsistency Management

### Definition
- checking for inconsistencies in software models
  - logic-based approach
  - model checking
  - specialised automated analysis
  - human-based exploration

### Diagnosis
- identification of source, cause and impact of an inconsistency

### Handling
- identification of possible actions, selection of action(s), execution of actions
Approach

• present a set of well-defined inconsistencies

• provide a lightweight formalisation of the UML

• investigate Description Logics (DLs) as a formalism for the definition, detection and handling of inconsistencies
  • syntax and semantics representation of the modelling language,
  • precise definitions of inconsistencies and inconsistency detection,
  • precise definitions of inconsistency resolutions and interactive inconsistency resolutions.
Outline

- Classification of Inconsistencies
- Description Logics
- From UML to DLs
- Inconsistency Detection
- Inconsistency Resolution
- Model Refactoring
- Contributions and Future Work
# Classification

<table>
<thead>
<tr>
<th>Specification</th>
<th>Behavioural</th>
<th>Structural</th>
</tr>
</thead>
<tbody>
<tr>
<td></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>Specification-Instance</td>
<td>specification incompatibility</td>
<td></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></td>
<td>observation behaviour</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instance</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>
<tr>
<td></td>
<td>invocation interaction</td>
<td></td>
</tr>
</tbody>
</table>

References:  
[VMSJ03], [SVJM04], [MVS04]
Behavioural Instance Inconsistencies

Invocation Inheritance Inconsistency [EE95]

Invocation inheritance consistency means that each sequence of calls invocable on the superclass can also be invoked on the subclass.
Definition 56  Given $c, c' \in C_M$ and generalisationOf$(c, c')$: 

A PSM $\pi_{c'} = (S', T', L', \rho', \Lambda')$ is invocation inheritance consistent with a PSM $\pi_c = (S, T, L, \rho, \Lambda)$ if and only if,

$\forall \mu : \text{valid}(\mu, \{\rho\}, \pi_c) \Rightarrow \text{valid}(\mu, \{\rho'\}, \pi_{c'})$ and for the PSM traces $\gamma$ corresponding to $\mu$ in $\pi_c$ and $\gamma'$ corresponding to $\mu$ in $\pi'_{c'}$, it holds that $\gamma = \gamma'_{S}$.

A SD $\delta'$ is invocation inheritance consistent with a SD $\delta$, if and only if,

$\forall O \in \text{contained}(\delta, \{c\}) \forall v_O \in \delta : (\exists O' \in \text{contained}(\delta', \{c'\}) \Rightarrow \exists v'_{O'} \in \delta' : v_O/v_{\text{rec}} = v'_{O'}/v_{\text{rec}})$.

A PSM $\pi_{c'} = (S', T', L', \rho', \Lambda')$ is invocation inheritance consistent with a SD $\delta$ if and only if,

$\forall O \in \text{contained}(\delta, \{c\}) \forall v_O/v_{\text{rec}} = < e_1 \ldots e_n > (\text{with } v_O \in \delta) \exists \mu_{c'} = < \tau_1 \ldots \tau_m > : (\forall \tau_i \in \mu_{c'} : \tau_i \in L' \land m \geq n \land \exists \sigma : \text{valid}(\mu_{c'}, \sigma, \pi_{c'}) \land \forall i \in \{1, \ldots, n\} \exists j \in \{i, \ldots, m\} : \tau_j = (op, g, h) = \text{label}(e_i) \land (\exists \tau_k = \text{label}(e_{i+1}) \in \mu_{c'} \Rightarrow k > j) \land \mu_{c'}, v_O/v_{\text{rec}} \text{ are in strict sequence})$. 
Outline

- Classification of Inconsistencies
- Description Logics
- From UML to DLs
- Inconsistency Detection
- Inconsistency Resolution
- Model Refactoring
- Contributions and Future Work
Why DL-based Approach?

- Logic-based approaches
  - general inconsistency rules
  - deduction
  - declarative semantics

- Description Logic-based approach
  - knowledge representation system
  - decidable
  - active research community
A PERSON is a HUMAN and HAS A GENDER that is FEMALE or MALE
FEMALE and MALE are disjoint
A WOMAN is a PERSON and HAS GENDER FEMALE

Only PARENTs can HAVE CHILDREN that are PERSONs
A TEENAGE-MOTHER is defined as MOTHER with max. age 20

Eve is a mother
Eve has a child Betty
Eve has a child Charles
Charles has a sibling Betty
Charles has at most one sibling
### Description Logics

**TBox**

\[
\begin{align*}
\text{PERSON} & \subseteq \text{HUMAN} \sqcap \exists \text{has-gender}.(\text{FEMALE} \sqcup \text{MALE}) \\
\text{FEMALE} & \subseteq \neg \text{MALE} \\
\text{WOMAN} & \subseteq \text{PERSON} \sqcap \exists \text{has-gender}.\text{FEMALE} \\
\text{TOP} & \subseteq \forall \text{has-child}.\text{PERSON} \\
\exists \text{has-child}.\text{TOP} & \subseteq \text{PARENT} \\
\text{TEENAGE-MOTHER} & \equiv \text{MOTHER} \sqcap (\text{max has-age} \ 20)
\end{align*}
\]

**ABox**

- Eve: MOTHER
- (Eve, Betty): has-child
- (Eve, Charles): has-child
- (Charles, Betty): has-sibling
- Charles: (at-most 1 has-sibling)
Tbox Standard Inferences

Contradictions in family Tbox?

Contradiction in a “woman” concept?

Is a Person a Human?

Tbox coherence

concept satisfiability

concept subsumption

classification of a Tbox
Abox Standard Inferences

Contradictions in family Abox w.r.t. family Tbox?

Is Eve a man?

All mothers in family Abox?

Direct type of Charles?

Get all siblings of Betty?

Relations between Charles and Alice?

Abox consistency

instance checking

retrieval

direct types of an individual

set of fillers of a role for an individual

set of roles between two individuals
## Description Logic Systems

<table>
<thead>
<tr>
<th>System</th>
<th>Language</th>
<th>CD</th>
<th>Abox</th>
<th>Query Language</th>
<th>Maintained?</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASSIC</td>
<td>$ALNF_{h^{-1}}$</td>
<td>no</td>
<td>yes</td>
<td>min</td>
<td>no</td>
</tr>
<tr>
<td>LOOM</td>
<td>$ALCQRIF_0$</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>FACT</td>
<td>$SHIQ$</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>RACER version 1.7</td>
<td>$SHIQ(D^-)$</td>
<td>yes</td>
<td>yes</td>
<td>min</td>
<td>yes</td>
</tr>
</tbody>
</table>
Outline

• Classification of Inconsistencies
• Description Logics
• From UML to DLs
• Inconsistency Detection
• Inconsistency Resolution
• Model Refactoring
• Contributions and Future Work
[VMSJ03], [VSM03], [VJM04], [Van04], [VMJ05]

Tbox representing UML metamodel

individual-of

Abox representing user-defined UML models

Class diagram

Traces

Call sequences

PSMs

spanning functions for UML PSM concepts

spanning functions for UML class diagram concepts

spanning functions for UML trace concepts

Tbox representing PSMs

Tbox representing call sequences

stimuli, links and slots

individual-of

[Ber02], [CCDL01]
- **receiving SD trace** of an instance \( o \) of a class \( c \) is a sequence of event occurrences \( <e_1, e_2, ..., e_n> \)
  - \( e_i = (m, \text{Cons, direction}), \text{direction} = "receive" \)
  - \( <\text{checkIfCashAvailable}, \text{dispenseCash}, \text{ejectCard}> \)
    - \( \text{ejectCard} = (m_e, \emptyset,"receive") \) \textbf{Invoked}(m_e) = \text{op}_{\text{EjectCard}} \)
- \textbf{instanceOf}(aCashDispensher, \{CashDispenser\}), \textbf{instanceOf}(aCardReader, \{CardReader\}), \textbf{instanceOf}(anATM, \{ATM\})
- \textbf{Links} = \{(aCashDispenser, aCardReader), (anATM, aSession), (anATM, aCardReader)\}
• **SD Encoding**
  
  • receiving SD trace
    $$<(m_1, \text{Cons}_1, \text{“receive”}), (m_2, \text{Cons}_2, \text{“receive”}), (m_3, \text{Cons}_3, \text{“receive”})>$$
  
  • subsumption and
  
  • a role $$r$$ used for sequencing
    
    • $$\text{cons'}_1 \cap m'_1 \subseteq \exists r. (m'_2 \cap \text{cons'}_2),$$
    
    • $$\text{cons'}_2 \cap m'_2 \subseteq \exists r. (m'_3 \cap \text{cons'}_3),$$
    
    where $$m' \models \forall \text{op}. \text{op'} \cap (= 1 \text{ op})$$
    
    • $$m_{\text{checkifcashavailable}} \subseteq \exists r. m_{\text{dispenseCash}}$$
    
    • $$m_{\text{dispenseCash}} \subseteq \exists r. m_{\text{ejectCard}}$$
A protocol state machine of a class $c$: $\pi_c = (S_c, T_c, L_c, \rho_c, \Lambda_c)$.

- $S_{ATM} = \{\text{PINEntry, AmountEntry, VerifyAccountBalance, VerifyATMBalance, GiveCash, ReturnReceipt, PrintReceipt, ReturnCard}\}$
- $T_{ATM} = \{\langle\text{PINEntry}\rangle, \text{getAmountEntry}, \{\text{AmountEntry}\}, \ldots\}$
- $L_{ATM} = \{\text{getAmountEntry, verifyAccount, \ldots}\}$

A call sequence is an $n$-tuple of labels in $L_c$, can be valid on a state configuration

- $\langle\text{dispenseCash, issueReceipt, ejectCard}\rangle$ is a call sequence on $\{\text{VerifyATMBalance}\}$
- $\text{dispenseCash} = (\text{opdispenseCash, \{balance >= amount\}, \emptyset})$
• **Call Sequence Encoding**

• Call sequence encoding is similar to SD trace encoding.

• Several call sequences?
  • `checkIfCashAvailable ⊑ ∃r.(ejectCard ⊔ dispenseCash)`,
    where, for example
    `ejectCard = ∀op.opEjectCard ⊓ (= 1 op) ⊓ <(balance, amount)`

• **Completeness of call sequences**
  • `∃r.X` replaced by `∀r.X`

• **Disjointness of operation calls**

• **State Information**

• Composite state
  • `Cash-Receipt ≜ GiveCash ⊔ ReturnReceipt ⊔ PrintReceipt`

• States in sequences
  • added to pre- and postconditions
• well-defined metamodel with a precise semantics
• interpretation of user-defined models as instances of metamodel
• conformance between metamodel and user-defined models
• semantics for class diagrams, sequence diagrams and PSMs
• interpretation of objects, links, stimuli, slots as instances of classes, associations, messages and attributes
Outline

• Classification of Inconsistencies
• Description Logics
• From UML to DLs
• Inconsistency Detection
• Inconsistency Resolution
• Model Refactoring
• Contributions and Future Work
# Classification Revisited

<table>
<thead>
<tr>
<th></th>
<th>Querying <em>ABoxes</em></th>
<th>Reasoning on <em>TBoxes</em></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Structural</strong></td>
<td>dangling type reference</td>
<td>incompatibility</td>
</tr>
<tr>
<td></td>
<td>inherited cyclic composition</td>
<td>specification behaviour</td>
</tr>
<tr>
<td></td>
<td>instance specification missing</td>
<td>incompatibility</td>
</tr>
<tr>
<td></td>
<td>disconnected model</td>
<td>invocation interaction</td>
</tr>
<tr>
<td></td>
<td></td>
<td>observation interaction</td>
</tr>
<tr>
<td></td>
<td></td>
<td>invocation behaviour</td>
</tr>
<tr>
<td></td>
<td></td>
<td>observation behaviour</td>
</tr>
<tr>
<td></td>
<td></td>
<td>invocation inheritance</td>
</tr>
<tr>
<td></td>
<td></td>
<td>observation inheritance</td>
</tr>
<tr>
<td></td>
<td></td>
<td>instance behaviour</td>
</tr>
<tr>
<td></td>
<td></td>
<td>incompatibility</td>
</tr>
<tr>
<td><strong>Behavioural</strong></td>
<td>multiplicity</td>
<td>specification behaviour</td>
</tr>
<tr>
<td></td>
<td>abstract object</td>
<td>incompatibility</td>
</tr>
<tr>
<td></td>
<td>navigation incompatibility</td>
<td>invocation interaction</td>
</tr>
<tr>
<td></td>
<td></td>
<td>observation interaction</td>
</tr>
<tr>
<td></td>
<td></td>
<td>invocation behaviour</td>
</tr>
<tr>
<td></td>
<td></td>
<td>observation behaviour</td>
</tr>
<tr>
<td></td>
<td></td>
<td>invocation inheritance</td>
</tr>
<tr>
<td></td>
<td></td>
<td>observation inheritance</td>
</tr>
<tr>
<td></td>
<td></td>
<td>instance behaviour</td>
</tr>
<tr>
<td></td>
<td></td>
<td>incompatibility</td>
</tr>
</tbody>
</table>

---

**Note:**
- *ABoxes* and *TBoxes* refer to different types of knowledge representation in classification theory.
- The table lists various structural and behavioural incompatibilities that can arise in classification frameworks.

- Declarative, decidable and sound queries of a (varying degree of) completeness.

- \textit{Behavioural inconsistencies}
  - detected by the \textit{Tbox} coherence task on DL \textit{Tboxes} representing SD traces and/or call sequences or PSMs.

\begin{verbatim}
\text{ans}(m, op, c)
\leftarrow \text{message}(m) \land \text{signature}(m, op) \land \text{operation}(op) \land \\
\text{receiveevent}(m, mend) \land \text{orderedasub}(mend, lifeline) \land \text{lifeline}(lifeline) \land \\
\text{represents}(lifeline, connectablecl) \land \text{base}(connectablecl, c) \land \text{class}(c) \land \\
\backslash (\text{ownedoperation}(c, op))) \land \backslash (\text{general}(superc, c))) \land \\
\text{ownedoperation}(superc, op))
\end{verbatim}

\[\text{[VSM03], [HMSW04]}\]
Tool Support RACOoN

The user can select inconsistency checks.

Explanation of the inconsistency checks:

arises when a stimulus or event references an operation that does not exist in the corresponding class (or its ancestors).

Results of the inconsistency checks:

Dangling Feature Reference
The following occurrences of this consistency problem were found in the current model:
The class CustomerConsole does not know the operation readAccountNbr(), which is currently defined on stimulus M2111.
Outline

- Classification of Inconsistencies
- Description Logics
- From UML to DLs
- Inconsistency Detection
- **Inconsistency Resolution**
- Model Refactoring
- Contributions and Future Work
Challenges

Which resolution action??

New inconsistency introduced!!!

Grouping resolution actions?
Rule-Based DL Inconsistency Resolution Approach

**IF** inconsistency $X$ occurs in model $M$

**THEN** change $M$ so that $X$ is resolved

**Abox and Tbox rules**

**Condition:**
- check inconsistency
- extra information
- user interaction

**Conclusion:** resolution
- Abox resolution actions: delete/add assertions
- Tbox resolution actions: retract/add axioms
Outline

- Classification of Inconsistencies
- Description Logics
- From UML to DLs
- Inconsistency Detection
- Inconsistency Resolution
- Model Refactoring
- Contributions and Future Work
Model Refactoring?

• A model refactoring is a transformation of a model into a different model expressed in the same modelling language to improve quality attributes of the model without changing the modelled (external) behaviour of the application.

• What is behaviour preservation?

• Source-code level
  • [Mens et al]: update, access and call preservation
Behaviour Preservation

![Diagram showing the concepts of inheritance and refactoring with consistency and preservation aspects.]

- **Inheritance**
  - **C** (base class)
  - **D** (derived class)
  - **C'** (modified class)
  - SD, PSM for different classes
  - Invocation/observation consistent

- **Refactoring**
  - **C**
  - **D** (new class)
  - observation/inheritance consistent
  - SD, PSM for different classes

- **Key Concepts**
  - Invocation call preservation
  - Observation inheritance consistency

---

Vrije Universiteit Brussel

System and Software Engineering Lab

38
Declare the method in the target class.

⇒ You may choose to use a different name, one that makes more sense in the target class.

Copy the code from the source method to the target. Adjust the method to make it work in its new home.

⇒ If the method uses its source, you need to determine how to reference the source object from the target method. If there is no mechanism in the target class, pass the source object reference to the new method as a parameter.

⇒ If the method includes exception handlers, decide which class should logically handle the exception. If the source class should be responsible, leave the handlers behind.

Compile the target class.

Determine how to reference the correct target object from the source.

⇒ There may be an existing field or method that will give you the target. If not, see whether you can easily create a method that will do so. Failing that, you need to create a new field in the source that can store the target. This may be a permanent change, but you can also make it temporarily until you have refactored enough to remove it.

Turn the source method into a delegating method.

Compile and test.

Decide whether to remove the source method or retain it as a delegating method.

⇒ Leaving the source as a delegating method is easier if you have many references.
Model Refactorings:
- after each step inconsistencies can be introduced
- same inconsistencies in multiple refactorings
- same inconsistency resolution can occur multiple times

Reuse of Inconsistency Resolutions in and across model refactorings
Tool Support $RACOoN$

- Proof-of-concept tool support
  - prototype tool plugged into Poseidon
  - detection and resolution of a representative set of inconsistencies
  - representative set of model refactorings for demonstration purposes
  - user interface can improve
Tool Support RACOoN
Outline

- Classification of Inconsistencies
- Description Logics
- From UML to DLs
- Inconsistency Detection
- Inconsistency Resolution
- Model Refactoring
- Contributions and Future Work
Contributions (1)

• Lightweight formalisation of important UML fragment

• Classification of domain-independent inconsistencies

• Encoding of UML fragment and AS in DLs
Contributions (2)

- Detection of inconsistencies through DL reasoning tasks and query language

- Rule-based inconsistency resolution approach

- Investigation of behaviour preservation and application of rule-based inconsistency resolution approach in model refactoring
Future Work (1)

- Larger set of UML elements
  - for example, activity diagrams
- Validation on large-scale cases
  - empirical data
- Management of inconsistencies and inconsistency resolutions
  - learning techniques, predefined resolution alternatives
- Model Refactorings
Future Work (2)

• Extending and improving tool support
  • improving user interface (experiments with users)
  • “intelligent” editing of inconsistency resolutions

• Extensions to DLs and DL Systems
  • proper feedback
  • integration of expressive rules