Variability within
Modeling Language Definitions

María Victoria Cengarle¹, Hans Grönniger², Bernhard Rumpe²

¹Software and Systems Engineering,
TU München, Germany

²Lehrstuhl Informatik 3 (Software Engineering),
RWTH Aachen, Germany
Goal

- Develop an approach with tool support to completely, formally define modeling languages
  - one target: UML
  - but not exclusively: approach should work with arbitrary modeling languages based on objects

- Challenges that we address
  - obtain a precise but variable semantics (in UML terms, support semantic variation points)
  - provide a flexible tool support that can be used for various verification scenarios
  - support underspecified and incomplete models (in contrast to complete programs)
  - integrate semantics of multiple views, multiple models (of different types)
Constituents of a Modeling Language Definition

- User interacts with **concrete syntax** (textual, graphical, ...)
- Internal representation as **abstract syntax** (AST, metamodel) and context conditions
- **Minimal** abstract syntax [optional] (reduced set of constructs to ease code gen. or semantics definition)
- Well-understood, precise **semantic domain** \( S \)
- Explicit **semantic mapping** of syntactic constructs of language \( L \) to elements of semantic domain, \( \text{sem} : L \rightarrow \wp(S) \)
Variability

- Formally handle variability in a modeling language definition

- Why?
  - Modeling languages should be open to customization in a precise and controllable way
  - Language developers state possible syntactic or semantic variants
  - Users, implementers configure (i.e. choose some of the) variants
  - Improve definition of UML, currently “implementors may provide [...] informal feature support statements [...] for less precisely defined dimensions such as presentation options and semantic variation points”

- Prerequisite
  - Analyze and classify variability
Presentation Variability

- Variability not present in a minimal abstract syntax

- Presentation options (as in UML)
  - map to the same abstract syntax
  - alternative keywords, font size, color etc.

- Abbreviations (syntactic sugar)
  - enhance readability but can be expressed by more “primitive” constructs of the language
  - transform hierarchical Statechart to equivalent flat Statechart
Syntactic Variability

- Variability affecting a minimal abstract syntax (and hence interacts with semantics)

- Language parameters
  - leave open which language is, e.g., used to express constraints or actions

- Syntax constraints, extensions
  - Optional context conditions to rule out models syntactically

- Set of valid stereotypes
  - general principle of extending the syntax of a language, encode semantic variability
Semantic Variability

- Subdivided into
  - semantic domain variability
  - semantic mapping variability

- Semantic domain and mapping analogy
  - mapping: “configuration options of a code generator”
  - domain: “properties of underlying run-time system”

- System Model as our semantic domain
  - Single semantic domain for various kinds of object-based modeling languages
  - Characterizes object-based systems by declaring constituents (structure, behavior, interaction) and constraining their properties
  - Necessarily of a certain complexity, but built in a modular way, introducing a hierarchy of theories using basic maths
System Model – high level overview

- Transition system $\Delta : \text{STATE} \rightarrow \wp(\text{STATE})$
  - closed world, non-deterministic, timed or untimed variants

- States $\text{STATE} = \text{DataStore} \times \text{ControlStore} \times \text{EventStore}$
  - data state: attribute values of objects
  - control state: active threads and computational state of methods
  - event state: messages to be processed

- Static information given through underspecified universes, e.g.
  - UCLASS, UOID – universe of classes and object identifiers
  - UTYPE, UVAL – universes of types and values
  - UOPN – universe of operations
  - sub – sub class relation
Semantic Variability

- **Semantic domain variability**
  - system model contains variants (optional definitions)
  - different notions of type-safe method overriding, restriction of inheritance to single inheritance etc.

  \[
  \text{Variant SingleInheritance} \\
  \forall C_1, C_2, C_3 \in \text{UCLASS}. \\
  C_1 \text{ sub } C_2 \land C_1 \text{ sub } C_3 \Rightarrow C_2 \text{ sub } C_3 \lor C_3 \text{ sub } C_2
  \]

- **Semantic mapping variability**
  - alternative realization choices for mapping functions
  - mapping of stereotypes

  \[
  \text{Variant MapClassStereotypes} \\
  m\text{ClassStereotypes stereos cl} = \\
  \text{<<singleton>>} \in \text{stereos} \Rightarrow \#(\text{oids cl}) = 1
  \]
## Variability Classification Summary

<table>
<thead>
<tr>
<th>Presentation Variability</th>
<th>Variability Not Present in a Minimal Abstract Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Presentation Options</td>
<td>Affect Concrete Syntax Only</td>
</tr>
<tr>
<td>Abbreviations</td>
<td>Can Be Omitted Without Losing Expressiveness</td>
</tr>
<tr>
<td>Syntactic Variability</td>
<td>Variability Affecting a Minimal Abstract Syntax</td>
</tr>
<tr>
<td>Stereotypes</td>
<td>Syntactic Encoding of Semantic Variability</td>
</tr>
<tr>
<td>Language Parameters</td>
<td>Useable With Independent Languages</td>
</tr>
<tr>
<td>Syntax Constraints</td>
<td>Constrain the Set of Well-Formed Models</td>
</tr>
<tr>
<td>Semantic Variability</td>
<td>Variability in the Semantics</td>
</tr>
<tr>
<td>Semantic Domain Variability</td>
<td>Variability in the Underlying Target Domain</td>
</tr>
<tr>
<td>Semantic Mapping Variability</td>
<td>Different Choices for Mapping Functions</td>
</tr>
</tbody>
</table>
Capturing & Configuring Variability

- Document variants and their interdependencies which may occur between variants on all levels

- Configure a language to fix (some) variants, others left underspecified

- Examples:
  - stereotype requires semantic mapping that handles it
  - language parameter requires semantics mapping for selected language
  - restrict multiple inheritance syntactically by context condition or semantically in the system model or by using a delegate mechanism in the semantic mapping (choose one!)
  - exclusive choice between two mapping functions, system model variants
Feature Diagrams to model variability

Legend:
- optional feature
- alternative feature
- inclusion/exclusion constraint
- selected feature

Diagram:
- SystemModel
- vType
- vObject
- SingleInheritance
- vMapAssocs
- vMapSuperClasses
- mapSuperCDirect
- mapSuperCDelegate

excludes SystemModel. vObject.SingleInheritance
Overview Tool Support

- **Possible benefits**
  - machine-readable and checkable semantics
  - directly suitable for various verification scenarios
  - control and quality check artifacts

- **Focus on**
  - textual modeling languages
    (although conceptually metamodeling would work, too)
  - keeping tool-support almost as flexible as “pencil and paper”

- **Tools**
  - MontiCore:
    - framework for the modular syntax definition of textual modeling languages (feat. context-free grammars, embedding, inheritance)
  - Isabelle/HOL:
    - theorem prover with higher order logic
  - Textual feature diagrams and configurations
Overview Tool Support
(for semantic mapping and domain, syntax similar)

- System model and its variants formalized in Isabelle/HOL (once)
- MontiCore translates language grammar to Isabelle/HOL data type
  - abstract syntax as a deep embedding
- Semantic mapping maps abstract syntax to predicates over systems of the system model
- Feature Diagrams captures variants of the semantic mapping and domain
Example Semantic Domain and Mapping Configuration

configuration CDCfg {
  SystemModel.vObject.SingleInheritance,
  CDSemantics.vMapSuperclasses.MapSuperCDelegate
}

multiple system model configurations are merged into one

Generate integrated System Model theory

Generate semantic mapping

theory SystemModel imports SystemModel_base
Imports SingleInheritance
Imports MapSuperCDelegate

theory CDSemantics imports CDSemantics_base
Imports MapSuperCDelegate
Verification Scenarios

- Various verification scenarios possible
  - Deep embedding allows reasoning about
    - syntax, syntactic operators
    - semantic mapping
  - Translator for concrete models for reasoning about properties of these models

- Case studies
  - “Integrated semantics of two class diagrams is empty due to circular inheritance relationship”
  - “A state described by an object diagram violates an OCL invariant in a class diagram”
  - “The sequence of messages implied by a sequence diagram may not occur according to a Statechart specification”
Consistency (in the presence of variability)
- e.g., contradicting mapping functions such that
  \[ \forall m_1, m_2 . \text{sem}_1(m_1) \cap \text{sem}_2(m_2) = \emptyset \]
- inconsistencies in the system model formalization so
  \[ \{\text{sm} | \text{valid sm}\} = \emptyset \]
- prove that this is not so

Automation for proofs
- currently only manually conducted proofs
- investigate potential for generating helpful lemmas

Performance
- esp. with highly recursive type definitions, performance deteriorates
- languages up to the size of (almost) full Java can be handled
Conclusion

- Classification of variability that may be found in a modeling language definition

- Flexible tool support for complete, System model-based language definition with MontiCore, feature diagrams and Isabelle/HOL
  - support for language variants
  - machine-checkable
  - accessible to verification scenarios and quality control
  - support incomplete, multiple views, multiple models types

- Future Work on
  - elaborating feature model for UML
  - how to efficiently prove configurations consistent and enhance proof automation
- Thank you for your attention.