Variability within Modeling Language Definitions

María Victoria Cengarle[1], Hans Grönniger[2], Bernhard Rumpe[2]

[1]Software and Systems Engineering,
TU München, Germany

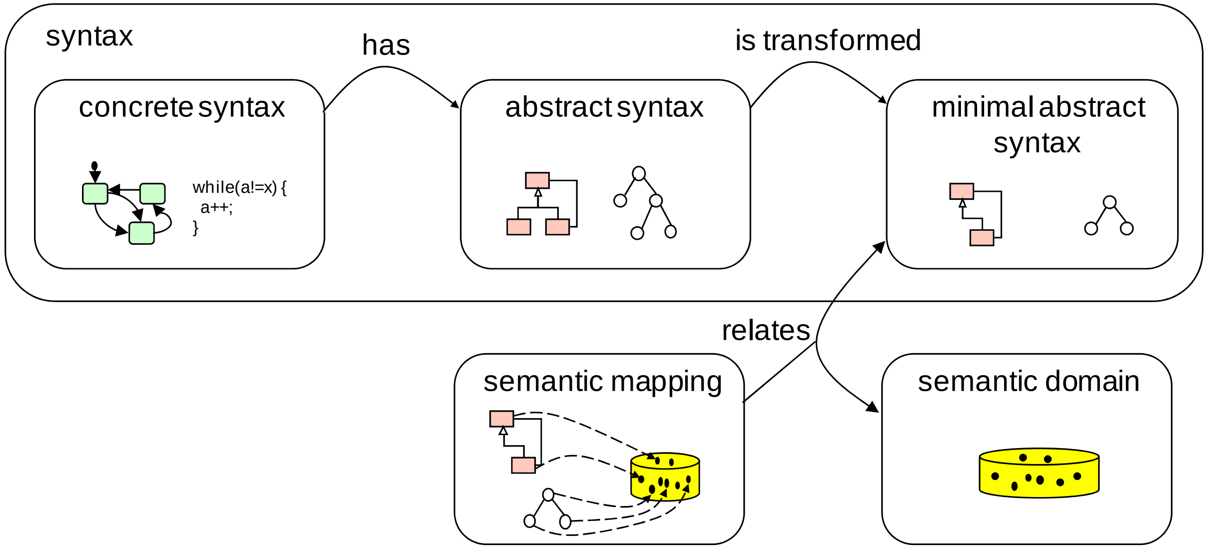
[2]Lehrstuhl Informatik 3 (Software Engineering),
RWTH Aachen, Germany

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, sem : $L \rightarrow \wp(S)$

Variability

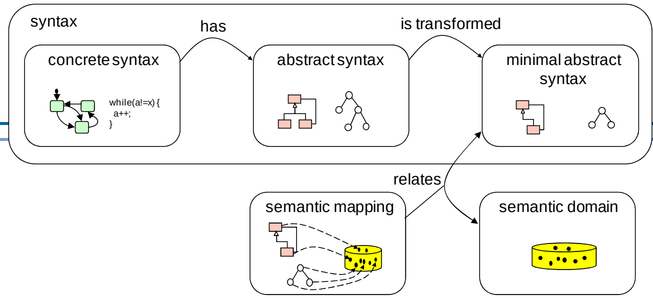
- Formally handle variability in a modeling language definition

- 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*"

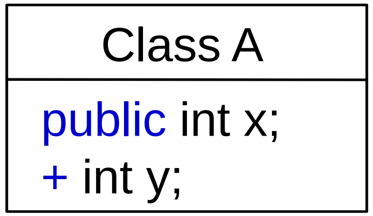
- Prerequisite
  - Analyze and classify variability

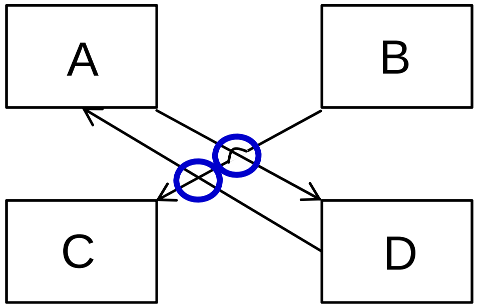
Hans Grönniger
Lehrstuhl für
Software Engineering
RWTH Aachen

Presentation Variability

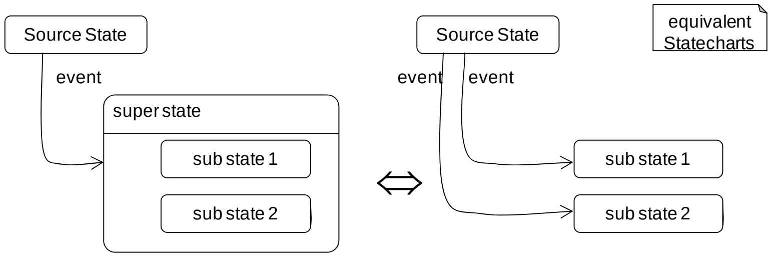


- Variability not present in a minimal abstract syntax
- Presentation options (as in UML)
  - map to the same abstract syntax
  - alternative keywords, font size, color etc.



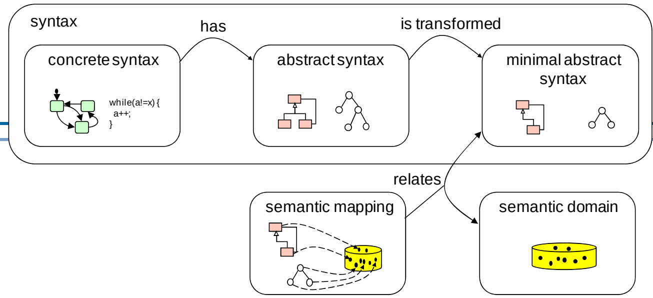


- Abbreviations (syntactic sugar)
  - enhance readability but can be expressed by more “primitive” constructs of the language
  - transform hierarchical Statechart to equivalent flat Statechart



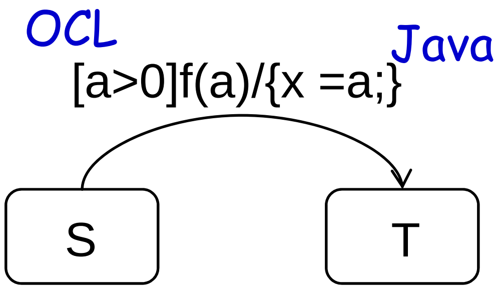
Hans Grönniger
Lehrstuhl für
Software Engineering
RWTH Aachen

Syntactic Variability

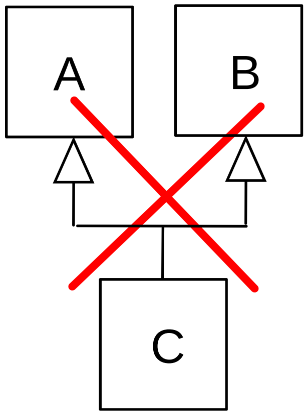


- Variability affecting a minimal abstract syntax (and hence interacts with semantics)

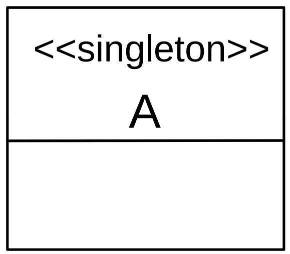
- Language parameters
  - leave open which language is, e.g., used to express constraints or actions



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



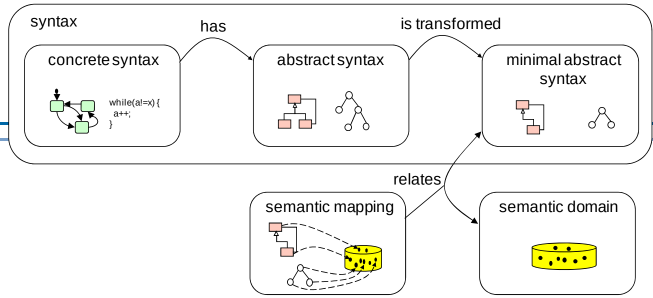
- Set of valid stereotypes
  - general principle of extending the syntax of a language, encode semantic variability



Hans Grönniger
Lehrstuhl für
Software Engineering

RWTH Aachen

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

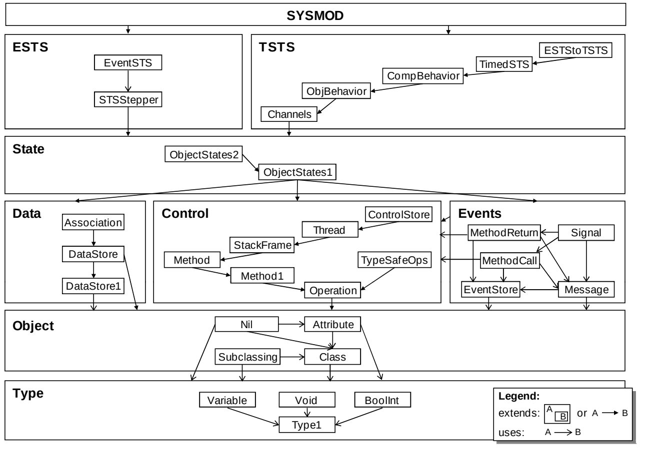
Hans Grönniger
Lehrstuhl für
Software Engineering
RWTH Aachen

System Model – high level overview

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

- States STATE = DataStore $\times$ ControlStore $\times$ 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.

```
Variant SingleInheritance
 ∀ C1, C2, C3 ∈ UCLASS.
    C1 sub C2 ∧ C1 sub C3 ⇒ C2 sub C3 ∨ C3 sub C2
```

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

```
Variant MapClassStereotypes
 mClassStereotypes stereos cl =
   <<singleton>> ∈ stereos ⇒ #(oids cl) = 1
```

Hans Grönniger
Lehrstuhl für
Software Engineering
RWTH Aachen

Variability Classification Summary

presentation variability	variability not present in a minimal abstract syntax
presentation options	affect concrete syntax only
abbreviations	can be omitted without losing expressiveness
syntactic variability	variability affecting a minimal abstract syntax
stereotypes	syntactic encoding of semantic variability
language parameters	useable with independent languages
syntax constraints	constrain the set of well-formed models
semantic variability	variability in the semantics
semantic domain variability	variability in the underlying target domain
semantic mapping variability	different choices for mapping functions

Capturing & Configuring Variability

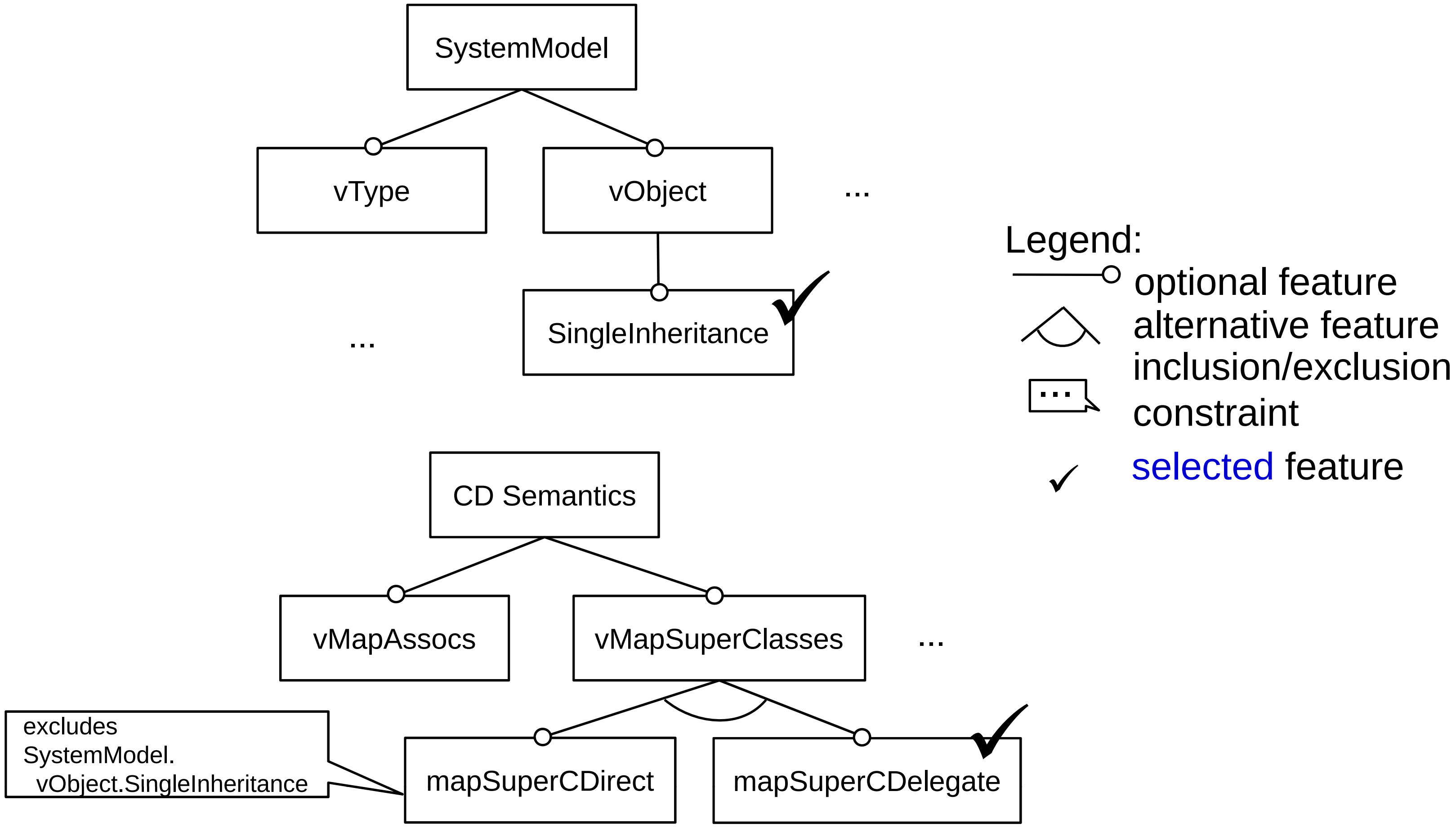
- Document variants and their interdependencies which may occur between variants on all levels

- 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

Capturing & Configuring Variability

- Feature Diagrams to model variability



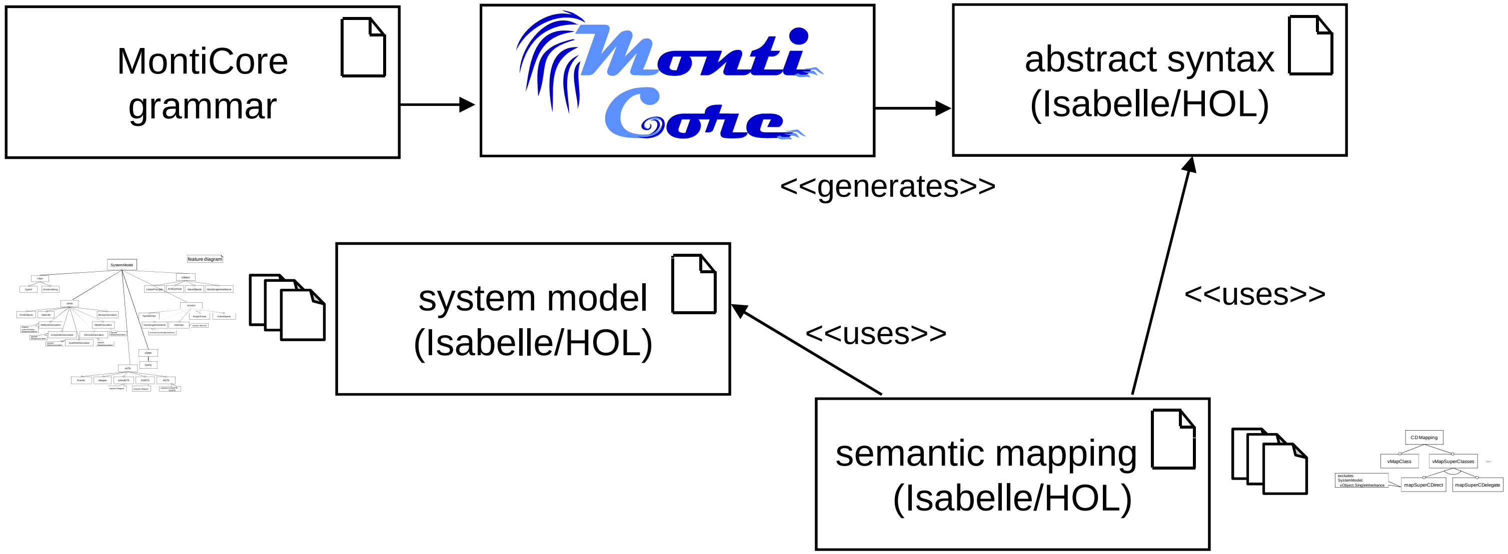
Overview Tool Support

- 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

Hans Grönniger
Lehrstuhl für
Software Engineering
RWTH Aachen

Overview Tool Support

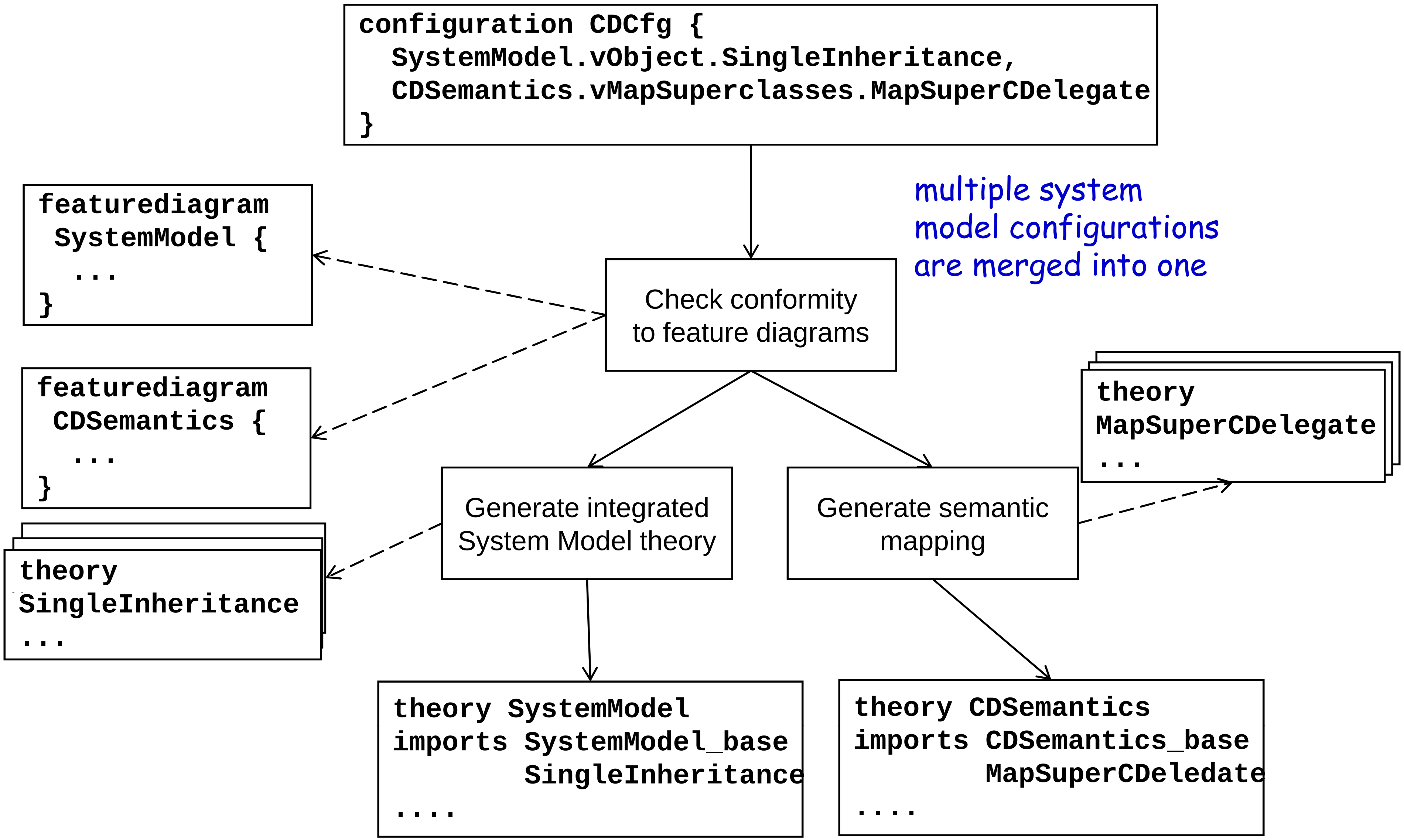
(for semantic mapping and domain, syntax similar)



- 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

Hans Grönniger
Lehrstuhl für
Software Engineering
RWTH Aachen

Example Semantic Domain and Mapping Configuration



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"

Issues

- Consistency (in the presence of variability)
  - e.g., contradicting mapping functions such that
    $\forall\, m_1, m_2 \,.\, sem_1(m_1) \cap sem_2(m_2) = \emptyset$
  - inconsistencies in the system model formalization so
    $\{sm \mid \text{valid } sm\} = \emptyset$
  - 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.