

Formal Analysis of UML/OCL Models

Achim D. Brucker

SAP RESEARCH

Vincenz-Priessnitz-Str. 1, 76131 Karlsruhe, Germany
achim.brucker@sap.com

University Bremen
Computer Science Colloquium
Bremen, 29th October 2008

Outline

- 1 Introduction
- 2 OCL in an Industrial Context
- 3 HOL-OCL
- 4 Mechanized Support for Model Analysis Methods
- 5 Conclusion and Future Work

Outline

- 1 Introduction
- 2 OCL in an Industrial Context
- 3 HOL-OCL
- 4 Mechanized Support for Model Analysis Methods
- 5 Conclusion and Future Work

The Situation Today

A Software Engineering Problem

- Software systems
 - are becoming more and more complex and
 - are used in safety and security critical applications.
- Formal methods are one way to increase their reliability.
- But, formal methods are hardly used by mainstream industry:
 - difficult to understand notation
 - lack of tool support
 - high costs
- Semi-formal methods, especially UML,
 - are widely used in industry, but
 - they lack support for formal methodologies.

Is OCL an Answer?

UML/OCL attracts the practitioners:

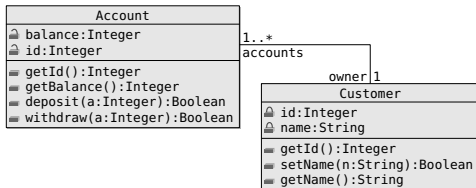
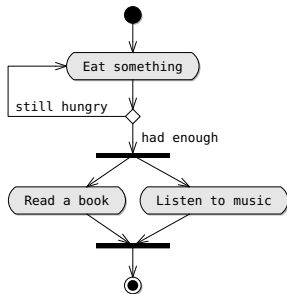
- is defined by the object-oriented community,
- has a “programming language face,”
- increasing tool support.

UML/OCL is attractive to researchers:

- defines a “core language” for object-oriented modeling,
- provides good target for object-oriented semantics research,
- offers the chance for bringing formal methods closer to industry.

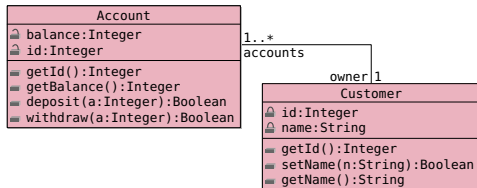
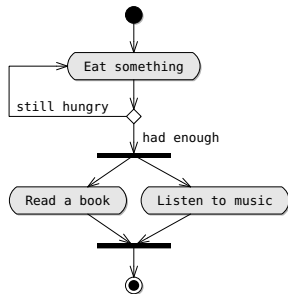
The Unified Modeling Language (UML)

- Visual modeling language
- Object-oriented development
- Industrial tool support
- OMG standard
- Many diagram types, e. g.,
 - activity diagrams
 - class diagrams
 - ...



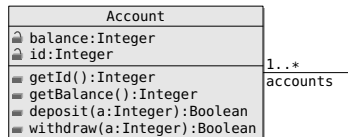
The Unified Modeling Language (UML)

- Visual modeling language
- Object-oriented development
- Industrial tool support
- OMG standard
- Many diagram types, e. g.,
 - activity diagrams
 - **class diagrams**
 - ...



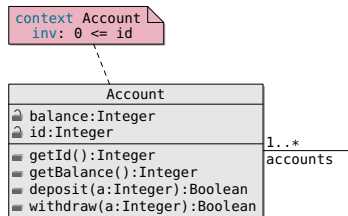
The Object Constraint Language (OCL)

- Textual extension of the UML
- Allows for annotating UML diagrams
- In the context of class-diagrams:
 - invariants
 - preconditions
 - postconditions
- Can be used for other diagrams



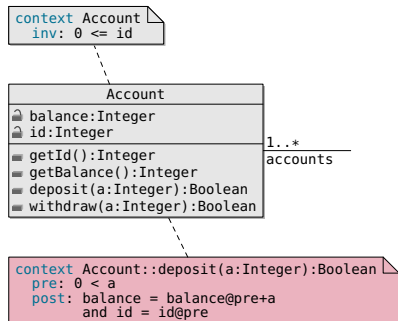
The Object Constraint Language (OCL)

- Textual extension of the UML
- Allows for annotating UML diagrams
- In the context of class-diagrams:
 - **invariants**
 - preconditions
 - postconditions
- Can be used for other diagrams



The Object Constraint Language (OCL)

- Textual extension of the UML
- Allows for annotating UML diagrams
- In the context of class-diagrams:
 - invariants
 - **preconditions**
 - **postconditions**
- Can be used for other diagrams



OCL by Example

- Class invariants:

```
context Account inv: 0 <= id
```

- Operation specifications:

```
context Account::deposit(a:Integer):Boolean
```

```
pre: 0 < a
```

```
post: balance = balance@pre + a
```

- A “uniqueness” constraint for the class Account:

```
context Account inv:
```

```
    Account::allInstances()
```

```
        ->forAll(a1,a2 | a1.id = a2.id implies a1 = a2)
```

OCL context

OCL keywords

UML path expressions

Outline

- 1 Introduction
- 2 OCL in an Industrial Context**
- 3 HOL-OCL
- 4 Mechanized Support for Model Analysis Methods
- 5 Conclusion and Future Work

Metamodeling and OCL (Revised)

- OCL can also be used to extend the MOF meta model
- 2.0 has a MOF-based metamodel for its abstract syntax
- OCL can be used for expressing queries on model content, e. g.,
 - model transformation implementation
 - event filtering

Level	MOF terms	OCL
M ₃	meta-meta-model	OCL specification
M ₂	meta-model	OCL constrains DSL
M ₁	model	OCL constrains model
M ₀	object	N/A

Target Groups and Impact: a Rough Picture

Level	MOF terms	OCL
M ₃	1	standards developer
M ₂	10 ... 100	tool developer
M ₁	1 000 ... 10 000	application developer
M ₀	100 000 ... 10 000 000	end user

Target Groups and Impact: a Rough Picture

Level	MOF terms	OCL
M ₃	1	standards developer
M₂	10 ... 100	tool developer
M ₁	1 000 ... 10 000	application developer
M ₀	100 000 ... 10 000 000	end user

Industrial OCL Support: An Example

Modeling Infrastructure (MOIN) developed by SAP:

- platform for SAP's next generation of modeling tools
- roughly similar to Eclipse (i.e., EMF), but not based on EMF
- provides an OCL 2.0 type checker
- provides an efficient evaluation environment (impact analysis for model changes)

At SAP, OCL is

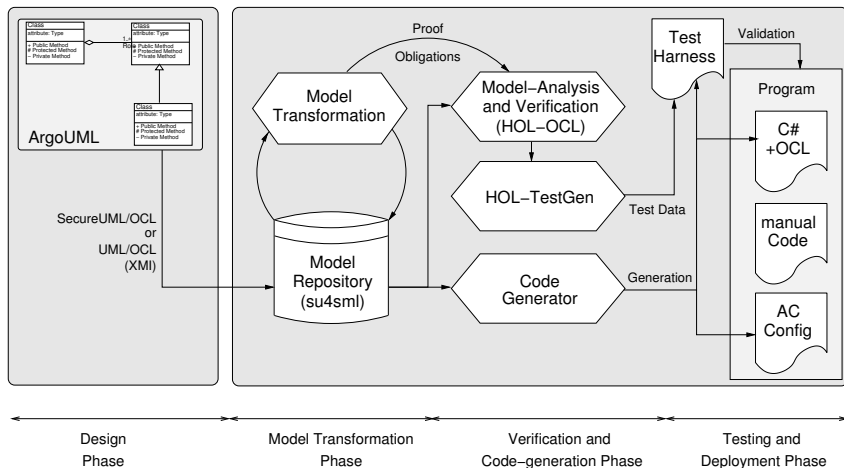
- widely used for annotating meta-models (M2)
- used by Development Architects

Outline

- 1 Introduction
- 2 OCL in an Industrial Context
- 3 HOL-OCL**
- 4 Mechanized Support for Model Analysis Methods
- 5 Conclusion and Future Work

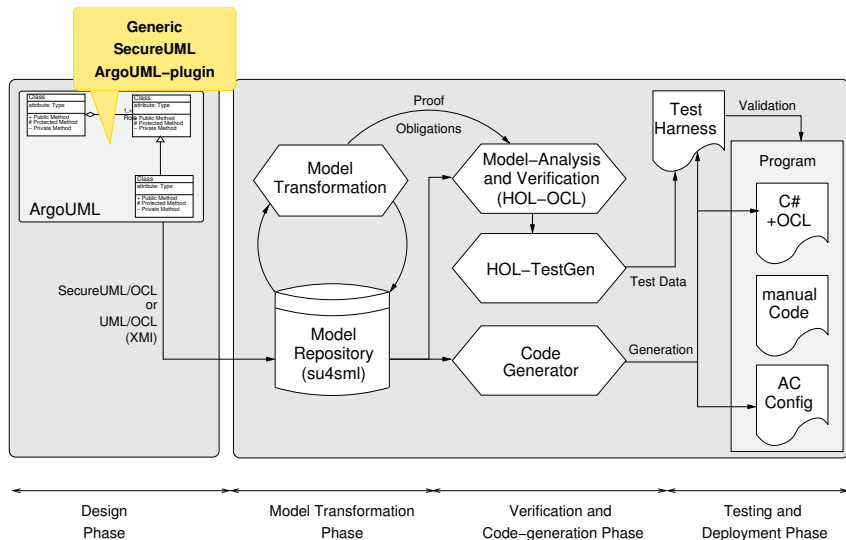
The HOL-OCL Vision:

Tool Supported Formal Methods for (Model-driven) Software Development



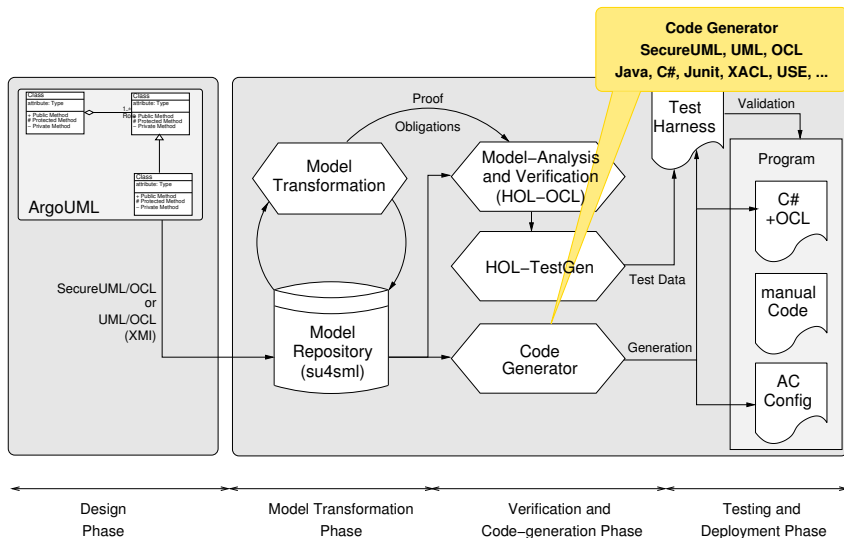
The HOL-OCL Vision:

Tool Supported Formal Methods for (Model-driven) Software Development



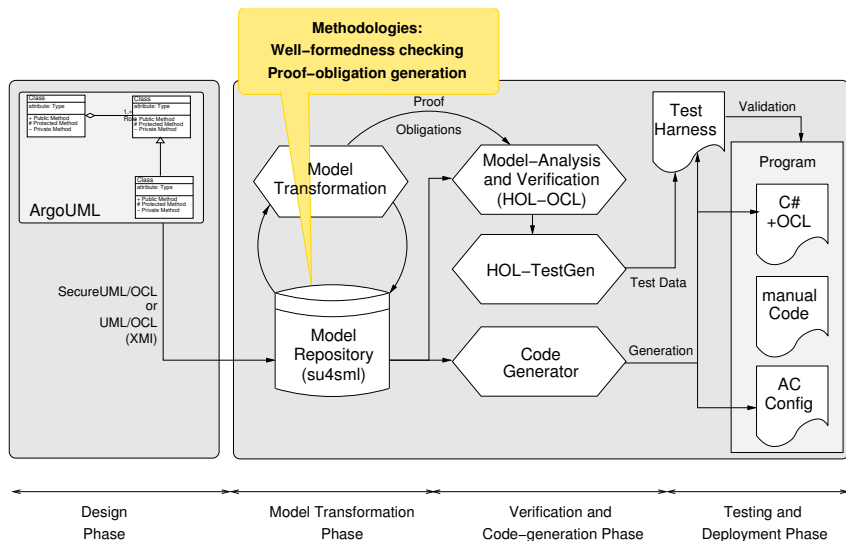
The HOL-OCL Vision:

Tool Supported Formal Methods for (Model-driven) Software Development



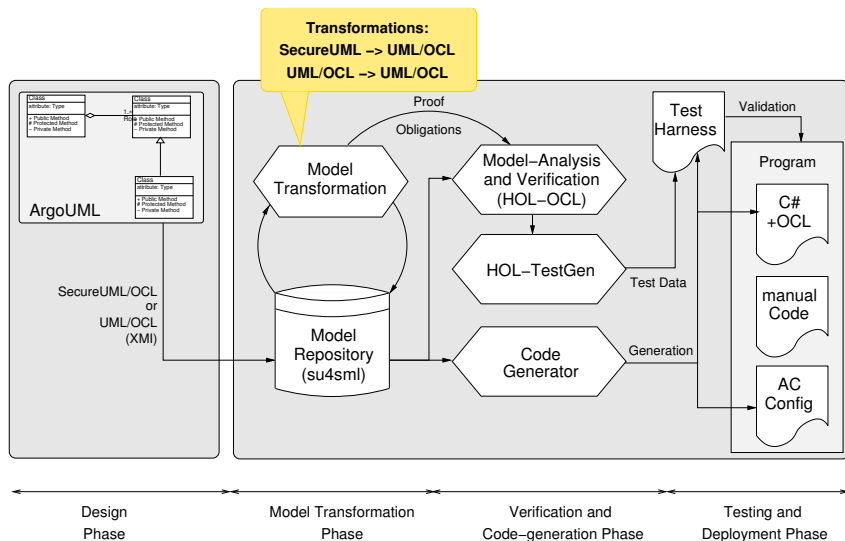
The HOL-OCL Vision:

Tool Supported Formal Methods for (Model-driven) Software Development



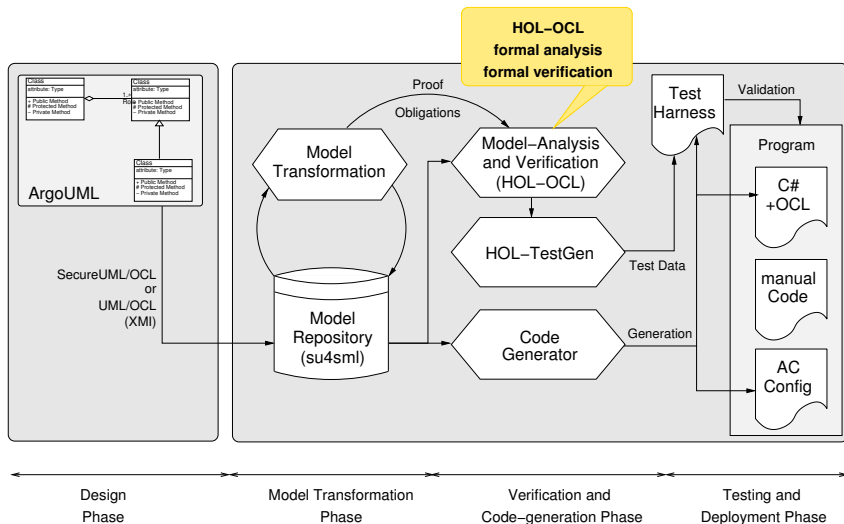
The HOL-OCL Vision:

Tool Supported Formal Methods for (Model-driven) Software Development



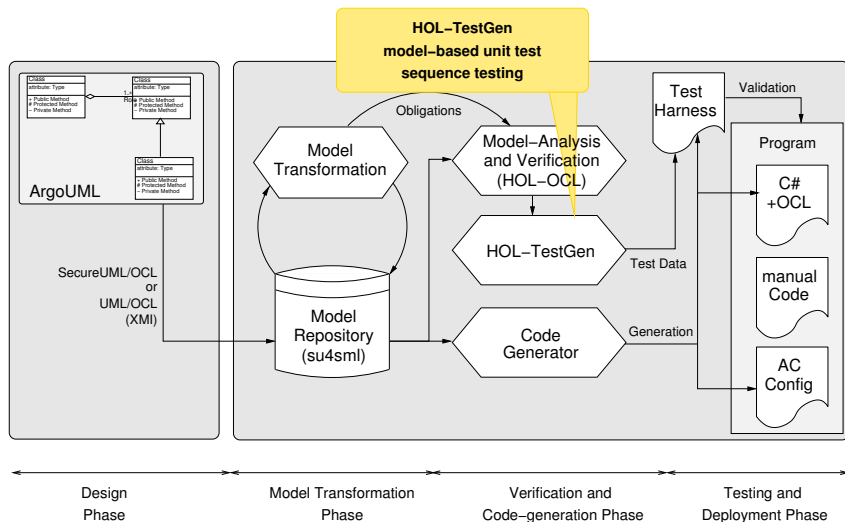
The HOL-OCL Vision:

Tool Supported Formal Methods for (Model-driven) Software Development



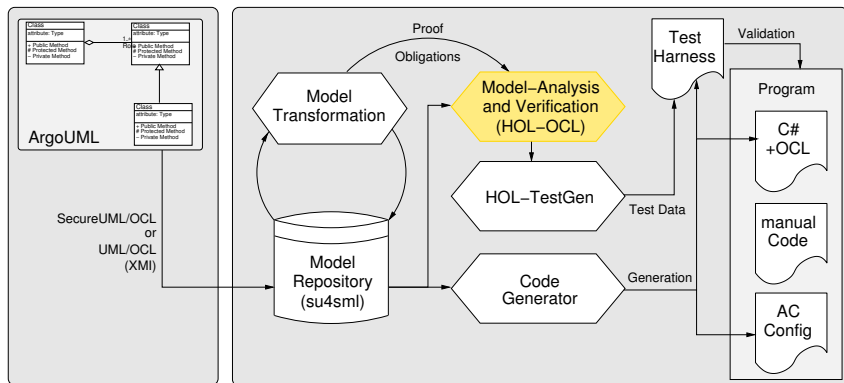
The HOL-OCL Vision:

Tool Supported Formal Methods for (Model-driven) Software Development



The HOL-OCL Vision:

Tool Supported Formal Methods for (Model-driven) Software Development



Design
Phase

Model Transformation
Phase

Verification and
Code-generation Phase

Testing and
Deployment Phase

HOL-OCL



- HOL-OCL provides:
 - a formal, machine-checked semantics for OO specifications,
 - an interactive proof environment for OO specifications,
 - publicly available:
<http://www.brucker.ch/projects/hol-ocl/>,
 - next (major) release planned in November 2008.
- HOL-OCL is integrated into a toolchain providing:
 - extended well-formedness checking,
 - proof-obligation generation,
 - methodology support for UML/OCL,
 - a transformation framework (including PO generation),
 - code generators,
 - support for SecureUML.

Outline

- 1 Introduction
- 2 OCL in an Industrial Context
- 3 HOL-OCL
- 4 Mechanized Support for Model Analysis Methods**
- 5 Conclusion and Future Work

Motivation

Observation:

- UML/OCL is a *generic* modeling language:
 - usually, only a sub-set of UML is used and
 - per se there is no standard UML-based development process.
- Successful use of UML usually comprises
 - a well-defined development process and
 - tools that integrate into the development process.

Conclusion:

- Formal methods for UML-based development should
 - support the local UML development methodologies and
 - integrate smoothly into the local toolchain.

A toolchain for formal methods should provide tool-support for **methodologies**.

Well-formedness of Models

Well-formedness Checking

- Enforce **syntactical** restriction on (valid) UML/OCL models.
- Ensure a minimal quality of models.
- Can be easily supported by fully-automatic tools.

Example

- There should be at maximum five inheritance levels.
- The Specification of public operations may only refer to public class members.
- ...

Proof Obligations for Models

Proof Obligation Generation

- Enforce **semantical** restriction on (valid) UML/OCL models.
- Build the basis for formal development methodologies.
- Require formal tools (theorem prover, model checker, etc).

Example

- Liskov's substitution principle.
- Model consistency
- Refinement.
- ...

Proof Obligations: Liskov's Substitution Principle

Liskov substitution principle

Let $q(x)$ be a property provable about objects x of type T . Then $q(y)$ should be true for objects y of type S where S is a subtype of T .

For constraint languages, like OCL, this boils down to:

- *pre-conditions* of overridden methods must be *weaker*.
- *post-conditions* of overridden methods must be *stronger*.

Which can formally expressed as implication:

- Weakening the pre-condition:

$$op_{pre} \rightarrow op_{pre}^{sub}$$

- Weakening the pre-condition:

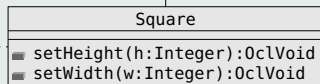
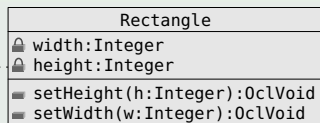
$$op_{post}^{sub} \rightarrow op_{post}$$

Proof Obligations: Liskov's Substitution Principle

Example

```
context Rectangle::setWidth(w:Integer):OclVoid
pre: w >= 0
post: self.width = w
```

```
context Square::setWidth(w:Integer):OclVoid
pre: w >= 0
post: self.width = w and self.height=w
```

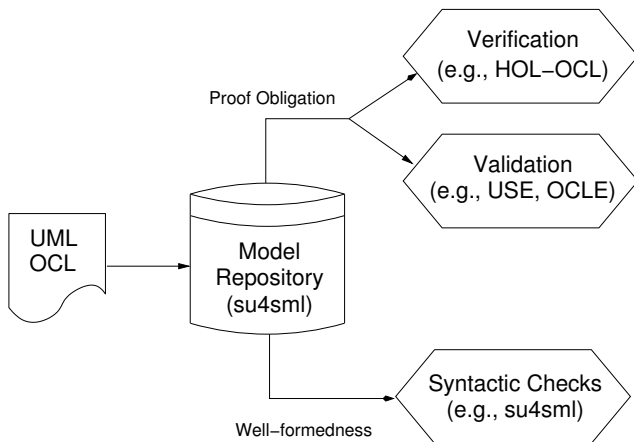


- Weakening the pre-condition:

$$(w \geq 0) \rightarrow (w \geq 0)$$

- Strengthening the post-condition:

Well-formedness and Proof Obligations



Methodology

A tool-supported methodology should

- integrate into existing toolchains and processes,
- provide a unified approach, integrating ,
 - syntactic requirements (well-formedness checks),
 - generation of proof obligations,
 - means for **verification** (proving) or **validation**, and of course
- all phases should be supported by tools.

Example

A package-based object-oriented refinement methodology.

Refinement – Motivation

Support top-down development from an abstract model to a more concrete one.

- We start with an abstract transition system

$$sys_{abs} = (\sigma_{abs}, init_{abs}, op_{abs})$$

- We refine each abstract operation op_{abs} to a more concrete one: op_{conc} .
- Resulting in a more concrete transition system

$$sys_{conc} = (\sigma_{conc}, init_{conc}, op_{conc})$$

- Such refinements can be chained:

$$sys_1 \rightsquigarrow sys_2 \rightsquigarrow \dots \rightsquigarrow sys_n$$

E.g., from an abstract model to one that supports code generation.

Refinement: Well-formedness

If package B refines a package A , then one should be able to substitute every usage of package A with package B .

- The concrete package must provide at a corresponding public class for each public class of the abstract model.
- For public attributes we require that their type and for public operations we require that the return type and their argument types are either basic datatypes or public classes.
- For each public class of the abstract package, we require that the corresponding concrete class provides at least
 - public attributes with the same name and
 - public operations with the same name.
- The types of corresponding abstract and concrete attributes and operations are compatible.

Refinement: Proof Obligations – Consistency

A transition system is consistent if:

- The set of initial states is non-empty, i. e.,

$$\exists \sigma. \sigma \in \mathit{init}$$

- The state invariant is satisfiable, i. e.,
the conjunction of all invariants is invariant-consistent:

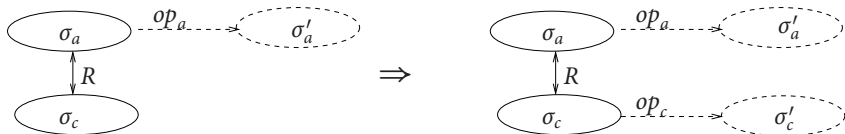
$$\exists \sigma. \sigma \models \mathit{inv}_1 \wedge \exists \sigma. \sigma \models \mathit{inv}_2 \wedge \dots \wedge \exists \sigma. \sigma \models \mathit{inv}_n$$

- All operations op are implementable, i. e.,
for each satisfying pre-state there exists a satisfying post-state:

$$\forall \sigma_{\text{pre}} \in \Sigma, \mathit{self}, i_1, \dots, i_n. \sigma_{\text{pre}} \models \mathit{pre}_{\text{op}} \longrightarrow \\ \exists \sigma_{\text{post}} \in \Sigma, \mathit{result}. (\sigma_{\text{pre}}, \sigma_{\text{post}}) \models \mathit{post}_{\text{op}}$$

Refinement: Proof Obligations – Implements

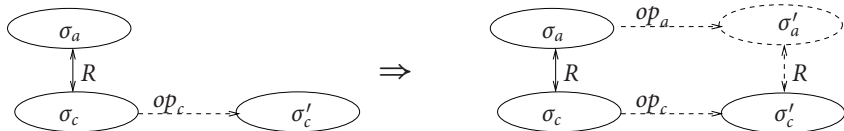
- Given an abstraction relation $R : \mathbb{P}(\sigma_{\text{abs}} \times \sigma_{\text{conc}})$ relating a concrete state S and an abstract states T .
- A forward refinement $S \sqsubseteq_{FS}^R T \equiv po_1(S, R, T) \wedge po_2(S, R, T)$ requires two proof obligations po_1 and po_2 .
- Preserve Implementability (po_1):**



$$po_1(S, R, T) \equiv \forall \sigma_a \in \text{pre}(S), \sigma_c \in V. (\sigma_a, \sigma_c) \in R \rightarrow \sigma_c \in \text{pre}(T)$$

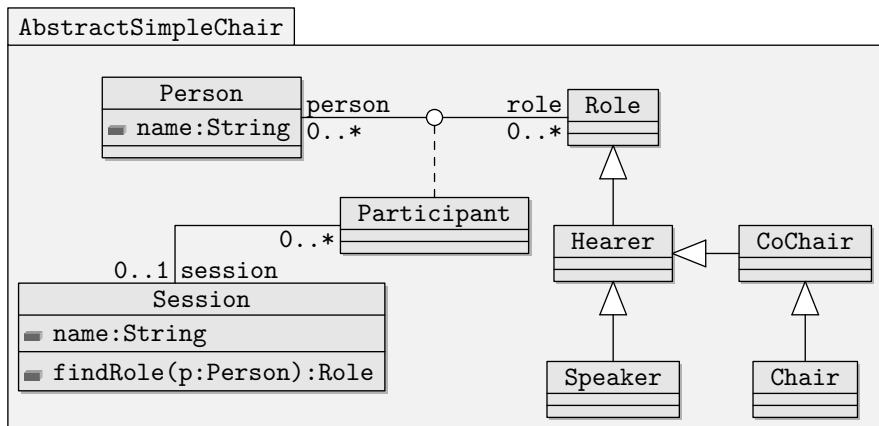
Refinement: Proof Obligations – Refines

- Given an abstraction relation $R : \mathbb{P}(\sigma_{\text{abs}} \times \sigma_{\text{conc}})$ relating a concrete state S and an abstract states T .
- A forward refinement $S \sqsubseteq_{FS}^R T \equiv po_1(S, R, T) \wedge po_2(S, R, T)$ requires two proof obligations po_1 and po_2 .
- Refinement (po_2):**

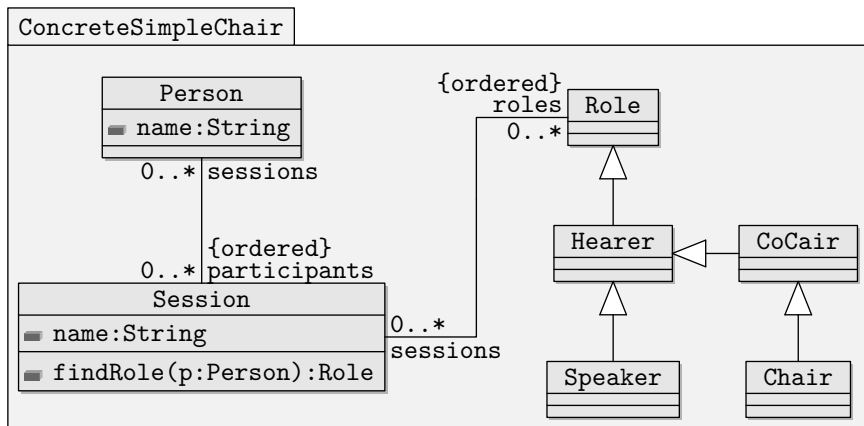


$$\begin{aligned}
 po_2(S, R, T) &\equiv \forall \sigma_a \in \text{pre}(S), \sigma_c \in V. \sigma'_c. (\sigma_a, \sigma_c) \in R \\
 &\quad \wedge (\sigma_c, \sigma'_c) \models_M T \rightarrow \exists \sigma'_a \in V. (\sigma_a, \sigma'_a) \models_M S \wedge (\sigma'_a, \sigma'_c) \in R
 \end{aligned}$$

Refinement Example: Abstract Model



Refinement Example: Concrete Model



Outline

- 1 Introduction
- 2 OCL in an Industrial Context
- 3 HOL-OCL
- 4 Mechanized Support for Model Analysis Methods
- 5 Conclusion and Future Work**

Ongoing and Future Work

- Ongoing work includes improving the infrastructures for
 - well-formedness-checking,
 - proof-obligation generation (Liskov, Refinement,),
 - consistency checking,
 - Hoare-style program verification,
 - better proof automation in general.
- Future works could include the development for
 - integrating OCL validation tools, e.g., USE,
 - test-case generation (i.e., integrating HOL-TestGen),
 - supporting SecureUML.
 -

Thank you
for your attention!

Any questions or remarks?

Bibliography I



Achim D. Brucker, Jürgen Doser, and Burkhart Wolff.
An MDA framework supporting OCL.
Electronic Communications of the EASST, 5, 2006.



Achim D. Brucker.
An Interactive Proof Environment for Object-oriented Specifications.
Ph.d. thesis, ETH Zurich, March 2007.
ETH Dissertation No. 17097.



Achim D. Brucker and Burkhart Wolff.
HOL-OCL – A Formal Proof Environment for UML/OCL.
In José Fiadeiro and Paola Inverardi, editors, *Fundamental Approaches to Software Engineering (FASE08)*, number 4961 in

Bibliography II

Lecture Notes in Computer Science, pages 97–100. Springer-Verlag, 2008.



Achim D. Brucker and Burkhart Wolff.

Extensible universes for object-oriented data models.

In Jan Vitek, editor, *ECOOP 2008 – Object-Oriented Programming*, number 5142 in Lecture Notes in Computer Science, pages 438–462. Springer-Verlag, 2008.



The HOL-OCL Website.

<http://www.brucker.ch/projects/hol-ocl/>.