Seminar: *Specification and Verification of Object-oriented Software*

The KeY Tool

developed by:
W. Ahrendt, T. Baar, B. Beckert, R. Bubel, M. Giese, R. Hähnle, W. Menzel,
W. Mostowski, A. Roth, S. Schlager, P.H. Schmitt, and others

Achim D. Brucker
Information Security, ETH Zürich, Switzerland
http://www.brucker.ch/
brucker@inf.ethz.ch

January 21, 2004
Wintersemester 2003/04

The Situation Today

- Software systems are
  - getting more and more complex.
  - used in safety and security critical applications.
- We think:
  - Complex software systems require a precise specification of its architecture and components.
  - Semi-formal methods (like UML diagrams) are not strong enough.

Specification should be useful, i.e. not only documentation!

Why use Formal Methods in Software Development

There are many reasons for using formal methods:

- safety critical applications, e.g. flight or railway control.
- security critical applications, e.g. access control.
- financial reasons (e.g. warranty), e.g. embedded devices.
- legal reasons, e.g. certifications.

Many successful applications of formal methods proof their success!

Why Formal Methods are not widely accepted in software industry?

- Only a few formal methods address industrial needs:
  - support for object-oriented programming.
  - highly automatic (?).
  - integration in standard CASE tools and processes.
- Formal methods people and industrial software developer are often speaking different languages.

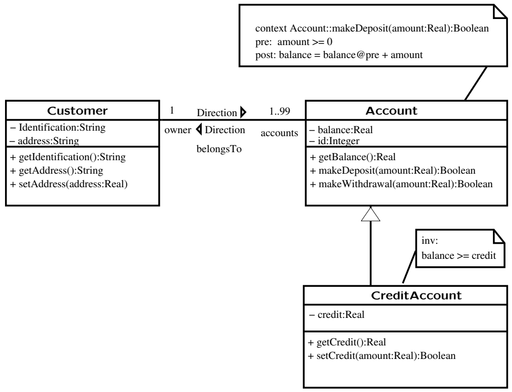
The KeY tool tackles these challenges by using a industrial accepted specification languages (UML/OCL) and by providing a strong integration into standard CASE tools.

Road Map

- Motivation
- Foundations: UML/OCL and JavaCard
- The KeY Tool
- Conclusion

UML

- diagrammatic OO modeling language
- many diagram types, e.g.
  - class diagrams (static)
  - state charts (dynamic)
  - use cases
- semantics currently standardized by the OMG
- wide use in SE-Tools (ArgoUML, Rational Rose, Together, . . .)

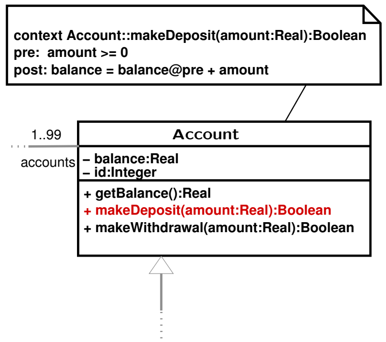


Are UML diagrams enough to specify OO systems formally?

- *The short answer:*
  - UML diagrams are not powerful enough for supporting formal reasoning over specifications.
- *The long answer:*
  We want to be able to
  - verify (proof) properties
  - refine specifications

OCL

- designed for annotating UML diagrams
- based on logic and set theory
- in the context of class–diagrams:
  - preconditions
  - postconditions
  - invariants
- will also be used for other diagram types
- part of the UML standard



OCL — A Simple Examples

- "Uniqueness" constraint for the class Account:

```
context Account inv:
Account.allInstances->forAll(a1,a2 | a1.id = a2.id implies a1 = a2)
```

- Properties of the class diagram can be described, e.g. multiplities:

```
context Account inv:
and Account.owner->size = 1
```

- Meaning of the method makeDeposit():

```
context Account::makeDeposit(amount:Real):Boolean
pre: amount >= 0
post: balance = balance@pre + amount
```

OCL keywords Syntax from UML model

Combining class diagrams and OCL results in a data-oriented formal specification language similar to Z, VDM, B,. . .

JavaCard

JavaCard is a proper subset of Java, excluding among other:

- threads
- dynamic class loading
- not all data types (e.g., float, double)
- restricted I/O (no GUI)

JavaCard supports basic object oriented features:

- state depends on local vars & attribute values of existing objects
- evaluation of expressions can have side effects
- int, short, arrays, reference types (aliasing)
- exceptions
- initialization of objects

The KeY Tool — Overview

KeY is a CASE tool extension for formal specifications, supporting

- the *creation* of constraints (design patters)
- the *formal analysis* of constraints
- the *verification* of implementations

These features are highly integrated into an commercial CASE tool, aiming for the

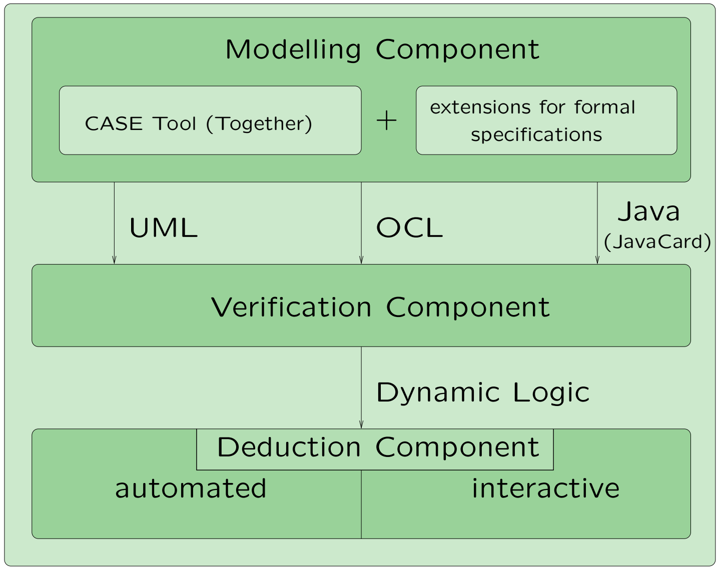
- development of industrial software without special needs for security.
- development of security critical software.
- use in education and training.

Excursion: Formal Challenges

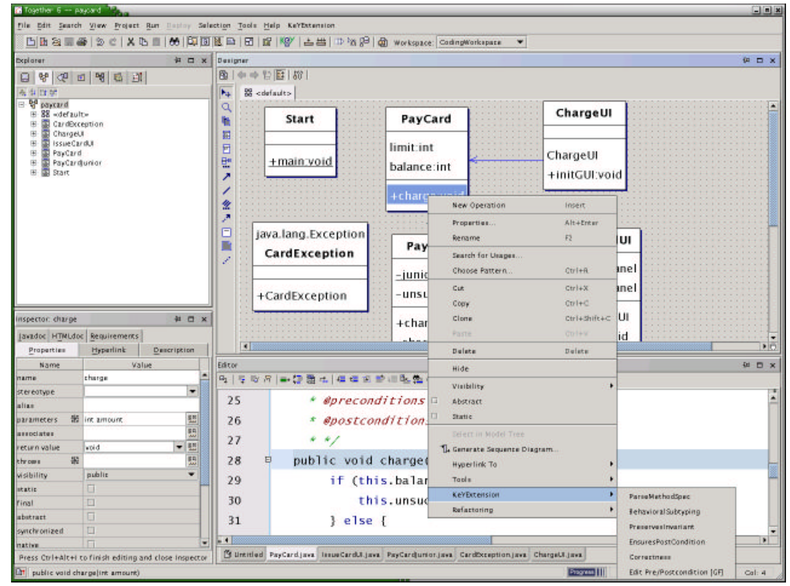
Only few formal methods are specialized for analyzing object oriented specifications.

- Problems and open questions:
  - object equality and aliasing
  - embedding of object structures into logics
  - referencing and dereferencing, including "null" references
  - dynamic binding
  - polymorphism
  - . . .
- Turning UML/OCL into a formal method:
  - semantics for OCL only given in a semi-formal way
  - OCL expressions are only meaningful together with the underlying UML model
  - no proof calculi for OCL
  - no refinement notions for OCL
  - . . .

The Architecture of the KeY tool



The Modelling Component



The Verification Component

- Translates OCL into dynamic logic (first order logic with modal operators)
- Generation of proof goals in first order logic, e.g. the invariant of a subclass implies the invariant of the superclass (Liskov).
- Generation of proof goals in dynamic logic, e.g. the implementation honors a given invariant.

```
context Account inv:
Account.allInstances->forAll(a1,a2 | a1.id = a2.id implies a1 = a2)
```

Can be translated into typed first order logic (using OO syntax):

$$(\forall a1, a2 : \mathrm{Account})(a1.\mathrm{id} \doteq a2.\mathrm{id} \rightarrow a1 \doteq a2)$$

Generated Proof Obligations

Example

- Proof obligation (OCL):

```
context Account inv:
Account.allInstances->forAll(a1,a2 | a1.id = a2.id implies a1 = a2)
```

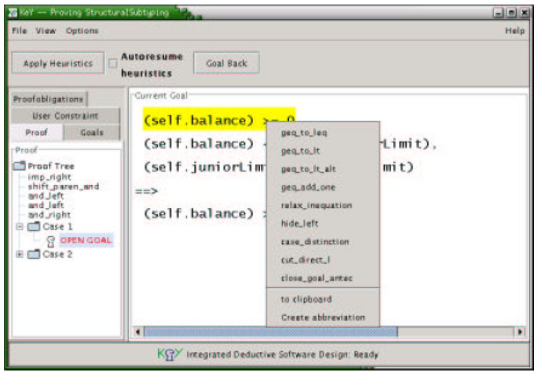
is an invariant of the method `getBalance()`

- Claim to be proven (dynamic logic):

$$(a1.\mathrm{id} \doteq a2.\mathrm{id} \rightarrow a1 \doteq a2) \vdash \langle \texttt{getBalance();} \rangle (a1.\mathrm{id} \doteq a2.\mathrm{id} \rightarrow a1 \doteq a2)$$

The Deduction Component

- based on a sequence calculus for dynamic logic
- automatic and interactive proof support
- counter example generation



The KeY Features

- create formal requirements specifications in OCL
- translate OCL requirements into correctness assertions in logic
- render OCL into natural language
- check correctness of a specification
- check correctness of a implementation
- generates counter examples for invalid assertions

Case Studies

The KeY tool was successfully applied to several larger case studies, e.g.:

Refinement: The Java Collection Framework (JFC)
Abstract specification of standard data structures (e.g. lists, sets) are stepwise refined to a concrete implementation.

Security: access control (PAM authentication with iButton)
Analysis of the state diagram of a JavaCard application used as authentication token.

Safety: computation of speed restrictions for Deutsche Bahn AG (railway)
A reference implementation computing a "speed book" was specified and analyzed (about 80 Java classes).

Further Work

- Support challenging Java Features (long term), e.g. threads, floating point arithmetic, dynamic class loading, GUI specification
- Integration of other formal techniques, e.g. model checking
- Integrate other UML diagram types, e.g. state charts
- Further applications, e.g. test case generation
- Build a formal, object-oriented software development process

Critics and Limitations

Nevertheless, there are some weak points of the KeY architecture:

- tactlets (tactics) are not shown to be correct.
- the semantic definitions (for OCL and JavaCard) are done in a axiomatic way and hence there is no guarantee for the consistency of the system.
- the translation from OCL into dynamic logic is quite naïve and doesn't honor the OCL standard. As such, KeY uses a "Dynamic Logic with OCL Syntax" instead of standard compliant OCL to annotate UML class diagrams.
- there is no guarantee that the axiomatized JavaCard semantics is compliant to Java standard.

Summary

- UML/OCL allows one to
  - formally specify object-oriented data models in prec-/postcondition style.
  - introduce formal methods in a lightweight way.
  - use an industry accepted OMG standard. This will hopefully lead to more acceptance (and hence tool support).
- The KeY tool allows one to
  - write formal OCL/UML specifications.
  - proof properties on the specification level.
  - proof properties on the implementation level.
  - proof that a implementation fulfills its specification.

  And all that providing an easy to use, first class integration into a widely accepted CASE tool!

The KeY tools fulfills its main goal, making formal methods usable by the object-oriented software industry.

Appendix

Slides for Answering Questions

A Program Logic: Dynamic Logic

- Syntax
  - typed first order logic
  - program logic
  - modal operators $[p]$ and $\langle p \rangle$
- Semantics
  - operators are evaluated in the terminating state of p
  - $[p]F$: if p terminates, then F holds (partial correctness)
  - $\langle p \rangle F$: p terminates and F holds (total correctness)

Calculus: "if-then" and "while" rule (simplified)

- if-then

$$\frac{pre, b \doteq \texttt{true} \vdash \langle p \rangle F \qquad pre, b \doteq \texttt{false} \vdash \langle q \rangle F}{pre \vdash \langle \texttt{if } b \texttt{ then } \{p\} \texttt{ else}\{q\} \rangle F}$$

- while

$$\frac{pre \vdash \langle \texttt{if } b \texttt{ then } \{p\} \texttt{ while}(b)\{p\} \rangle F}{pre \vdash \langle \texttt{while}(b)\{p\}\} \rangle F}$$

Related Work

- Tools supporting OCL can be roughly divided into:
  - Runtime checking of OCL constraints, e.g. based on the Dresden OCL compiler [7].
  - Model simulation and validation, e.g. the USE tool [9].
  - Proof environments, namely HOL-OCL [5]; which is implemented as a shallow embedding of OCL into Isabelle/HOL. HOL-OCL tries to strictly follow the OCL 2.0 standard, e.g. including a three valued logic.
- Formalizing the Java semantics:
  - μJava an embedding of a Java subset into Isabelle/HOL [8].

References

[1] The KeY project, January 2004. http://www.key-project.org.

[2] Wolfgang Ahrendt, Thomas Baar, Bernhard Beckert, Richard Bubel, Martin Giese, Reiner Hähnle, Wolfram Menzel, Wojciech Mostowski, Andreas Roth, Steffen Schlager, and Peter H. Schmitt. The key tool. Technical Report 2003-05, Chalmers, Göteborg University, 2003.

[3] Wolfgang Ahrendt, Thomas Baar, Bernhard Beckert, Martin Giese, Elmar Habermalz, Reiner Hähnle, Wolfram Menzel, and Peter H. Schmitt. The KeY approach: Integrating object oriented design and formal verification. Technical Report 2000/4, University of Karlsruhe, Department of Computer Science, January 2000. http://i12www.ira.uka.de/~projekt/publicat.htm.

[4] Achim D. Brucker and Burkhart Wolff. HOL-OCL: Experiences, consequences and design choices. In Jean-Marc Jézéquel, Heinrich Hussmann, and Stephen Cook, editors, *UML 2002: Model Engineering, Concepts and Tools*, number 2460 in Lecture Notes in Computer Science, pages 196–211. Springer-Verlag, Dresden, 2002. ISBN 3-540-44254-5. http://www.brucker.ch/bibliography/abstract/brucker.ea-hol-ocl-2002.

[5] Achim D. Brucker and Burkhart Wolff. A proposal for a formal OCL semantics in Isabelle/HOL. In César Muñoz, Sophiène Tahar, and Víctor Carreño, editors, *Theorem Proving in Higher Order Logics*, number 2410 in Lecture Notes in Computer Science, pages 99–114. Springer-Verlag, Hampton, VA, USA, 2002.

ISBN 3-540-44039-9. http://www.brucker.ch/bibliography/abstract/brucker.ea-proposal-2002.

[6] Richard Bubel and Reiner Hähnle. Integration of informal and formal development of object-oriented safty-critical software: A case study with the key system. In *Electronic Notes in Theoretical Computer Science*, volume 80. Elsevier Science Publishers, 2003.

[7] OCL Compiler Suite, 2001. http://dresden-ocl.sourceforge.net/.

[8] Tobias Nipkow, David von Oheimb, and Cornelia Pusch. μJava: Embedding a programming language in a theorem prover. In Friedrich L. Bauer and Ralf Steinbrüggen, editors, *Foundations of Secure Computation*, volume 175 of *NATO Science Series F: Computer and Systems Sciences*, pages 117–144. IOS Press, 2000. http://isabelle.IN.TUM.de/Bali/papers/MOD99.html.

[9] Mark Richters and Martin Gogolla. OCL - syntax, semantics and tools. In Tony Clark and Jos Warmer, editors, *Advances in Object Modelling with the OCL*, pages 43–69. Springer, Berlin, LNCS 2263, 2001.

Contents