CASE tool-based system development using UML/OCL

Achim D. Brucker
Institut für Informatik
Albert-Ludwigs-Universität Freiburg
brucker@informatik.uni-freiburg.de
http://www.informatik.uni-freiburg.de/~brucker

April 12, 2002

Motivation

☛ Why specify?

- Complex software systems require a precise specification of architecture and components.
- Semi-formal methods (like UML) are not strong enough.

☛ Why UML/OCL?

- UML is the standard modeling language in OO development.
- OCL is part of the OMG UML standard.

Specification should not only generate documentation!

Overview

1. The V-Model

2. UML/OCL

3. Using specifications: code generation, verification, validation,. . .

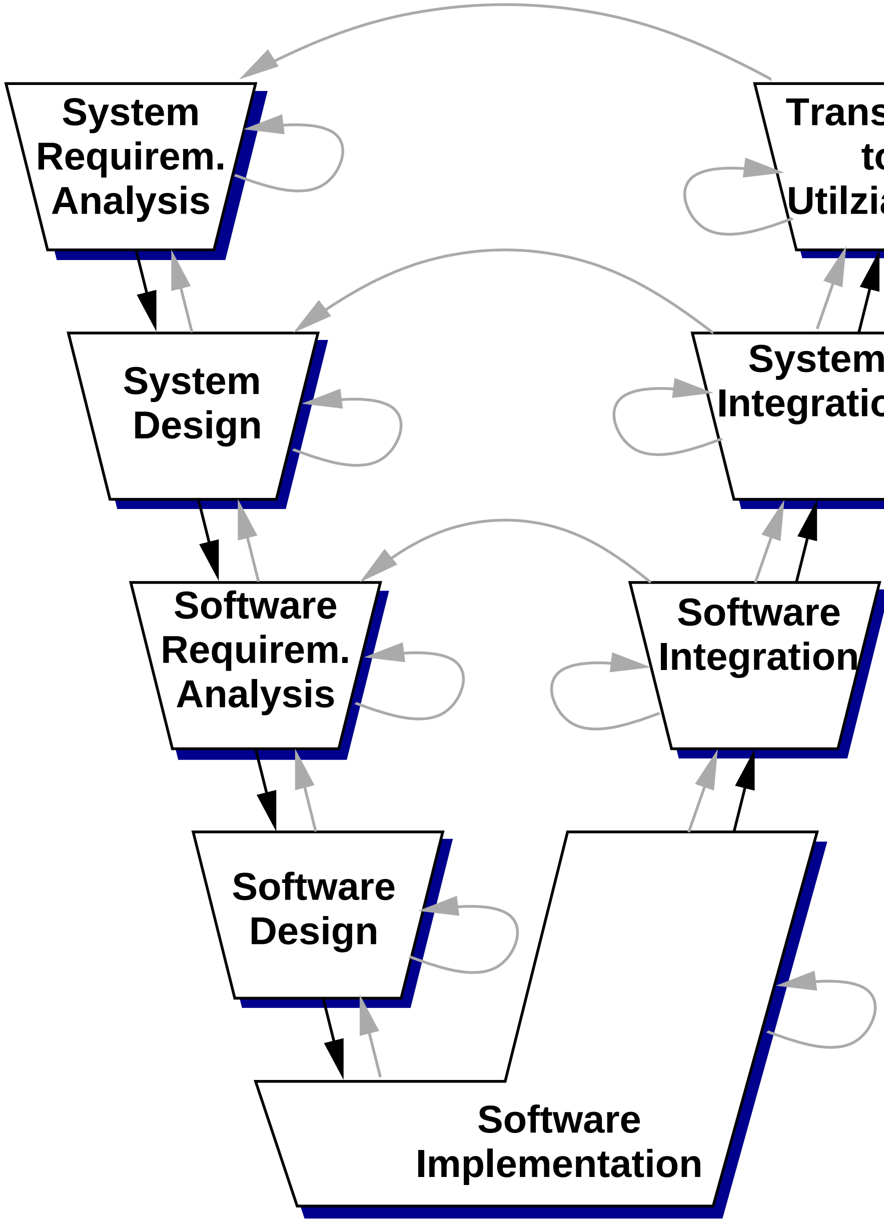
4. Two examples:

   - Automated test case generation using UML/OCL

   - ArcSecure

The V-Model (simplified)

- ☛ process and development model
- ☛ describes dependencies and (work) flows
- ☛ ISO standard
- ☛ an example of a phase-based development model



Benefits of using a (semi-) formal specification

☞ understanding and communication

☞ Formal reasoning and analysis (verification, model checking)

☞ generating code

☞ runtime assertion checking

☞ generation of test data for validation (testing)

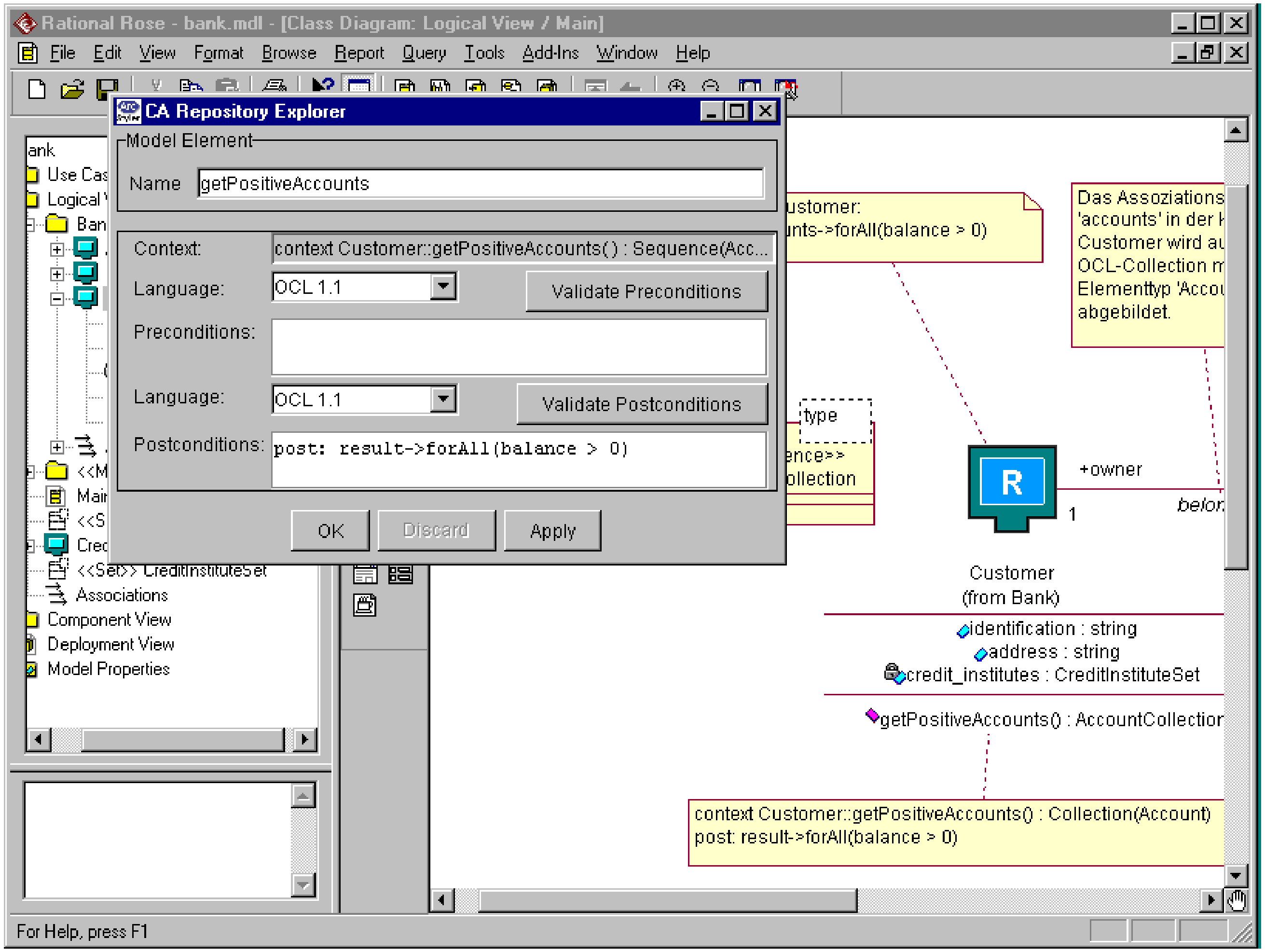
☞ use constraints for runtime assertion checking

☞ Documentation

CASE Tools

Computer **A**ided **S**oftware **E**ngineering tools support the software development process by providing a framework for:

- documentation
- specification
- code generation
- **validation**
- **verification**



Name: getPositiveAccounts

Context: context Customer::getPositiveAccounts() : Sequence(Acc...

Language: OCL 1.1

Preconditions:

Language: OCL 1.1

Postconditions: post: result->forAll(balance > 0)

ustomer:
unts->forAll(balance > 0)

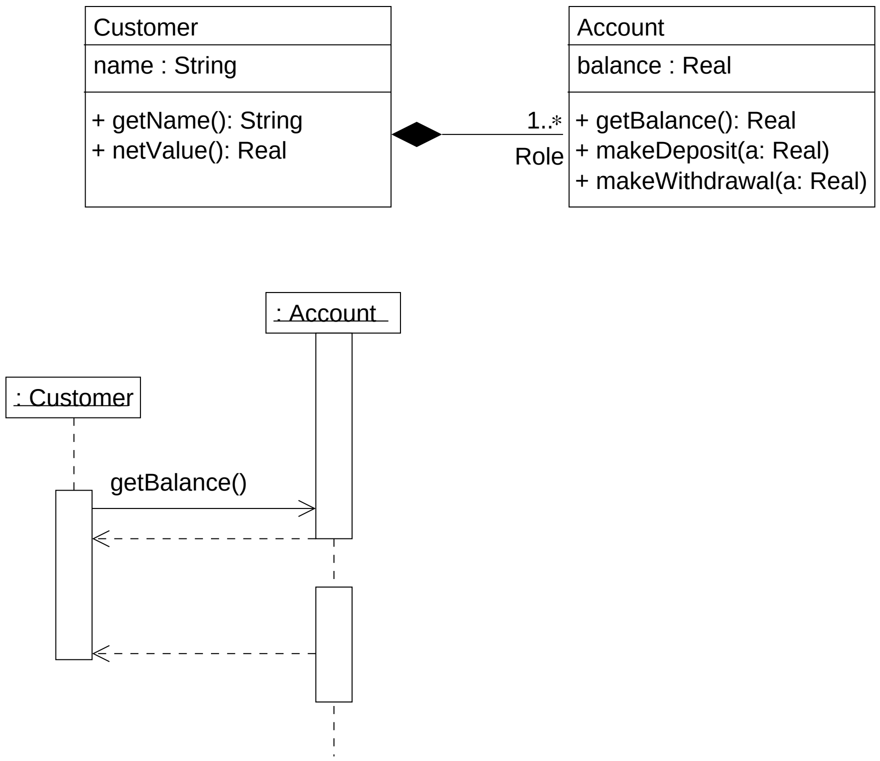
Das Assoziations
'accounts' in der k
Customer wird au
OCL-Collection m
Elementtyp 'Accou
abgebildet.

Customer
(from Bank)
identification : string
address : string
credit_institutes : CreditInstituteSet
getPositiveAccounts() : AccountCollection

context Customer::getPositiveAccounts() : Collection(Account)
post: result->forAll(balance > 0)

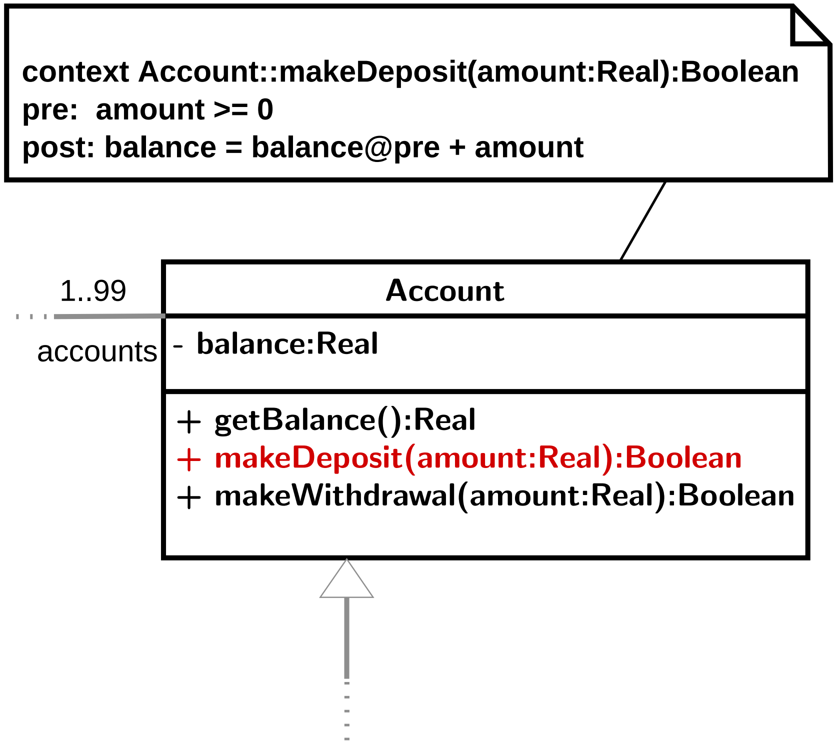
The Unified Modeling Language (UML)

- visual modeling language
- many diagram types, e.g.
  - class diagrams (static)
  - state charts (dynamic)
  - use cases
- diagrammatic method
- OO development
- OMG standard
- widely used



The Object Constraint Language (OCL)

- extension based on logic and set theory
- designed for annotating UML diagrams
- in the context of class–diagrams:
  - preconditions
  - postconditions
  - invariants
- can be used for other diagram

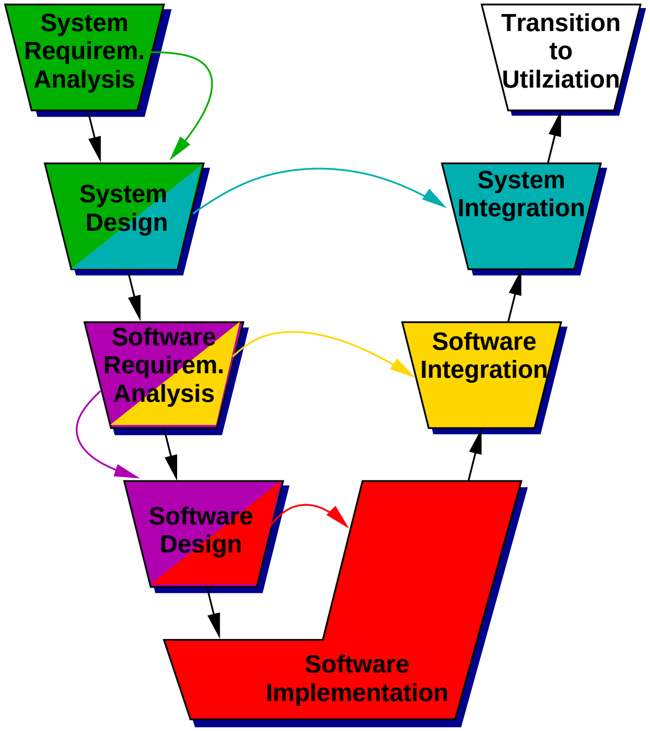


```
context Account::makeDeposit(amount:Real):Boolean
pre:  amount >= 0
post: balance = balance@pre + amount
```

1..99
accounts
Account
- balance:Real
+ getBalance():Real
+ makeDeposit(amount:Real):Boolean
+ makeWithdrawal(amount:Real):Boolean

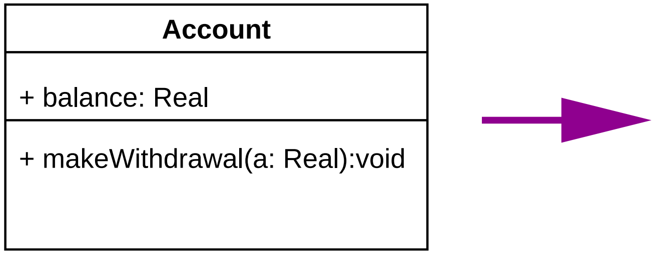
Verification and Model Checking

- ☞ prove that a implementation fulfills its specification
- ☞ *abstract:* prove properties of an abstract model
- ☞ *source code level:* prove properties of a concrete implementation
- ☞ often not fully automated
- ☞ needs a formal specification



Code Generation

- semi-formal: generate skeleton/stubs
- formal: generate implementation

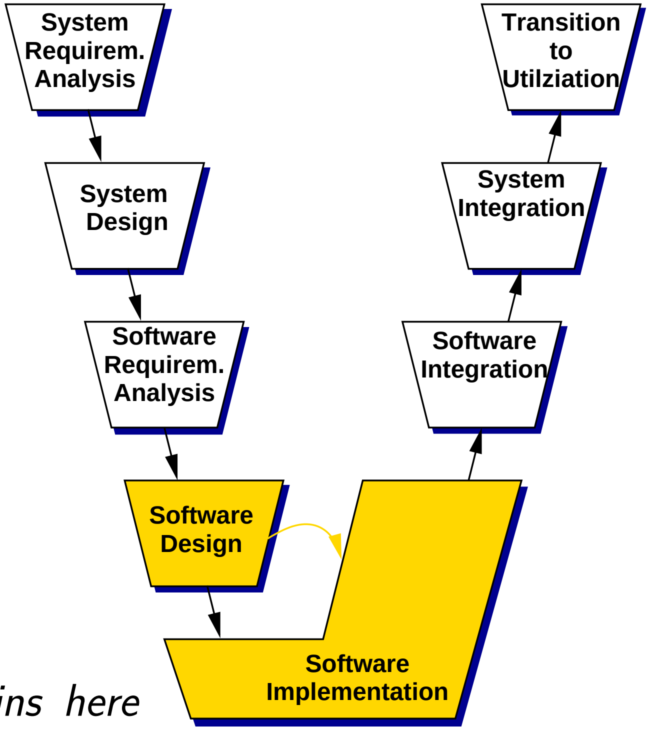


```
class Account{
        float balance;

        float getBalance(){
                return balance;
        }

        void setBalance(float balance){
                this.balance = balance;
        }

        void makeDeposit(float a){
                // user defined code begins here
                this.balance = this.balance + a;
                // end of user defined code
        }
}
```



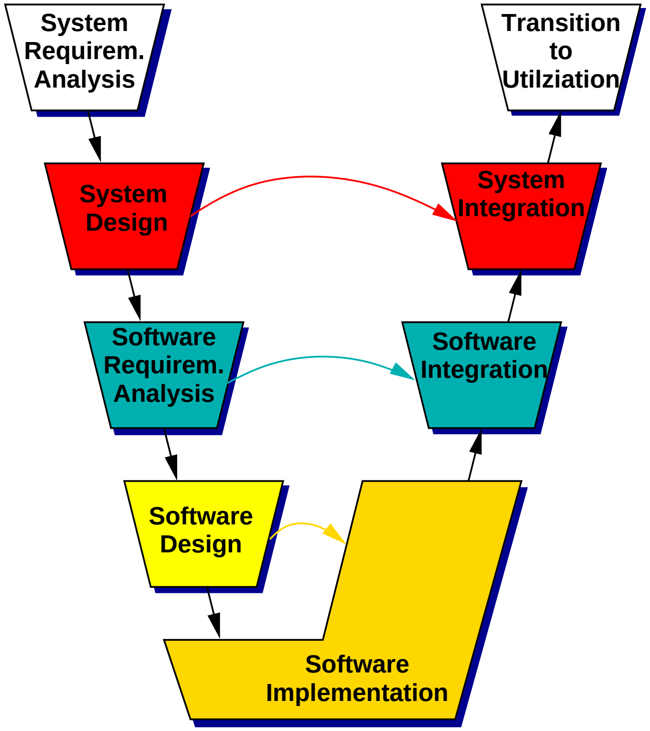
Assertion Checking

☛ generates runtime checks for constraints (pre-/post-conditions, invariants,. . .)

☛ slightly similar to `assert.h`

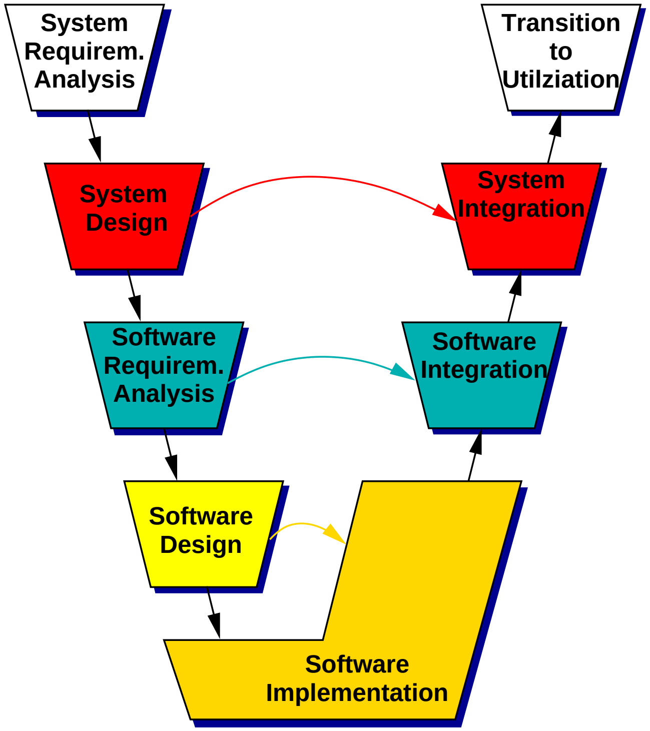
☛ a post-hoc debugging method

☛ needs a formal specification



Test Case Generation (Validation)

- test the implementation with a specified input
- validates the implementation against its specification
- meaningful testing requires high grade sets of test data
- no formal proof of correctness
- needs a formal specification



```
if ( (a  < 5) || ( a > 10) && (b=5)
{
   // Block A
}else{
   // Block B
}
```

Test Case Generation (Example)

Input: three integer, representing the length of the sides of a triangle

Output: whether the input describes an equilateral, isosceles, scalene or invalid triangle

Based on an OCL specification, it is possible to determine partition for test case selection automatically.

☞ already six partitions

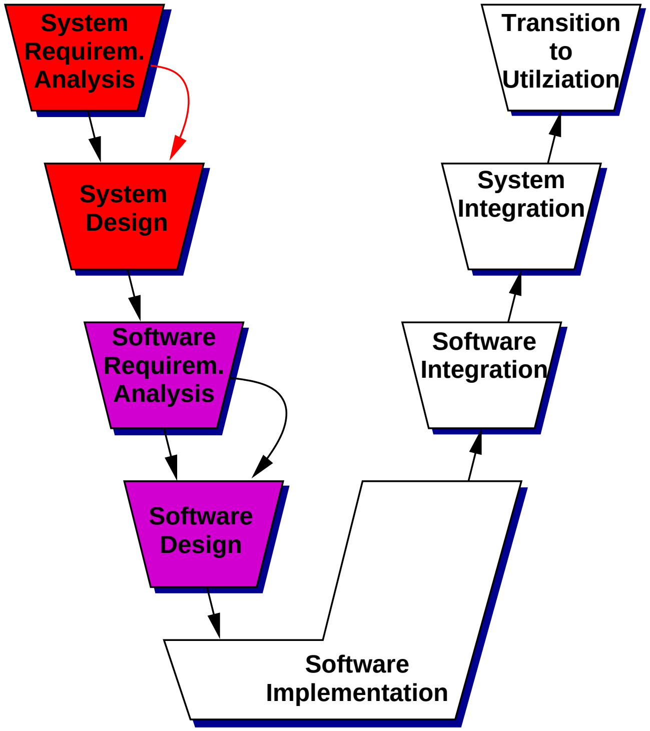
☞ select test cases from these partitions, exploiting boundary cases

Specifying Security (ArcSecure)

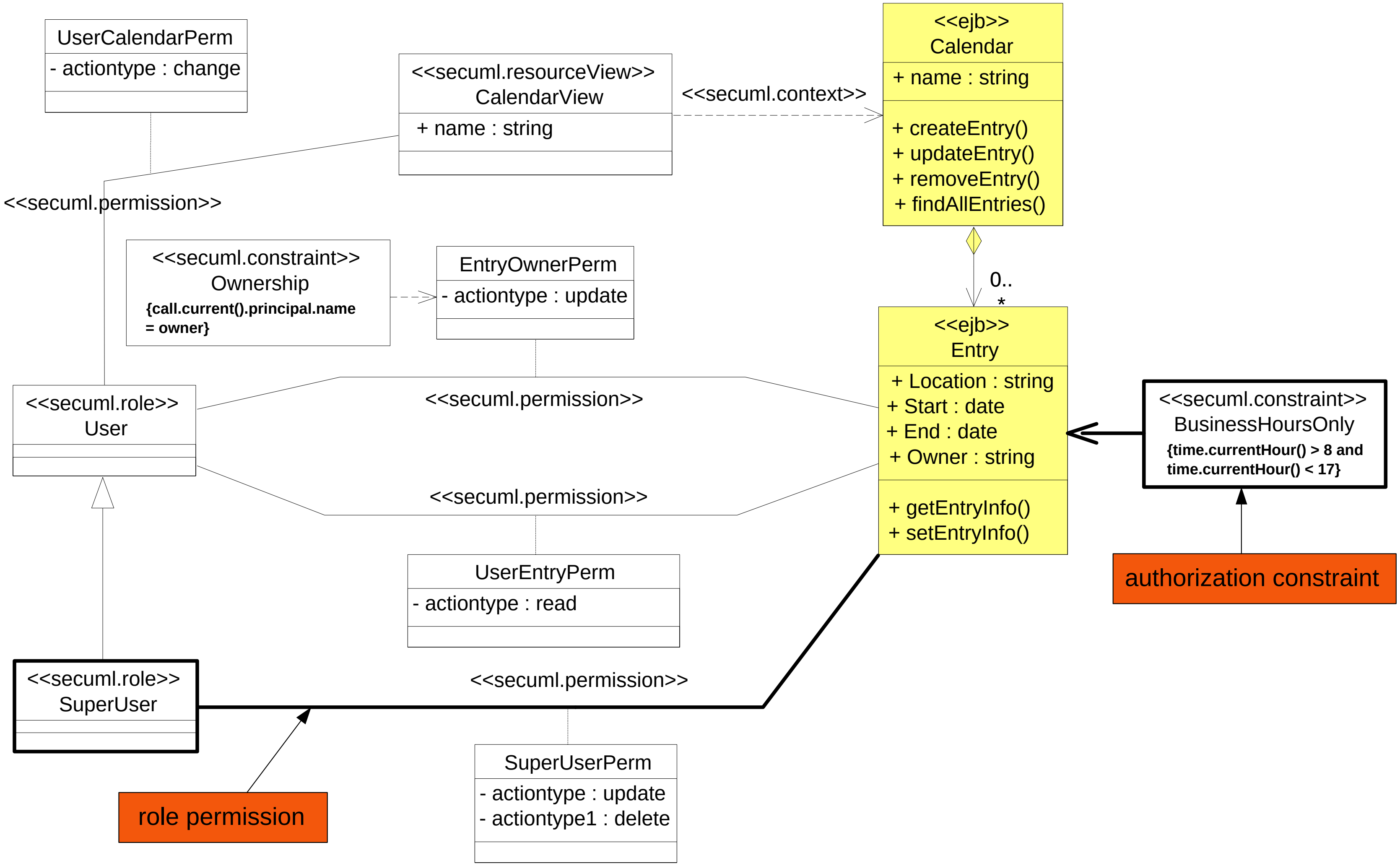
- model information needed for authorization
- based on RBAC with dynamic extensions
- code generation honors authorization constraints

- *only* for specification: informal possible
- further analysis requires semi-formal or formal specification

- ArcSecure can profit in all presented ways from the specification



Specifying Security (Example)



Conclusion

- Specification helps mastering complex projects
- Widely used CASE tools support:
  - documentation generation
  - code generation
  - assertion checking
- Specialized CASE tools and academia provide support for validation and verification.