Testing Distributed Component Based Systems Using UML/OCL

Achim D. Brucker and Burkhart Wolff
{brucker,wolff}@informatik.uni-freiburg.de

Albert-Ludwigs Universität – Freiburg, Germany
Interactive Objects GmbH – Freiburg, Germany

Integrating Diagrammatic and Formal Specification Techniques
Workshop at Informatik 2001 - 31. Jahrestagung der GI, ÖCG Jahrestagung 2001
September 26, 2001
Wien

Motivation

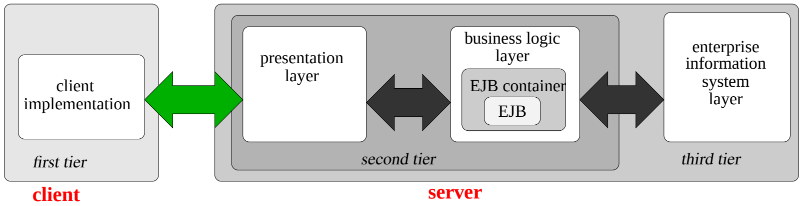
- Diagrammatic Methods raise interest:
  - Complex software systems increase the need for specification.
  - UML is the standard modeling language in industry.
- Middleware Architectures raise interest:
  - Distribution is the key–technology in the Internet.
  - Middleware offers possibilities to link new and legacy systems.
  - Well known middleware standards are CORBA and J2EE/EJB.

➤ *We explain the use of Diagrammatic Methods for modeling, specifying, and runtime testing of middleware architectures.*

Roadmap

1. Introduction
2. Specification of EJBs and Testing
3. Design Patterns for EJBs
4. Practical Experience
5. Future Work

Distributed Systems Using J2EE: Overview



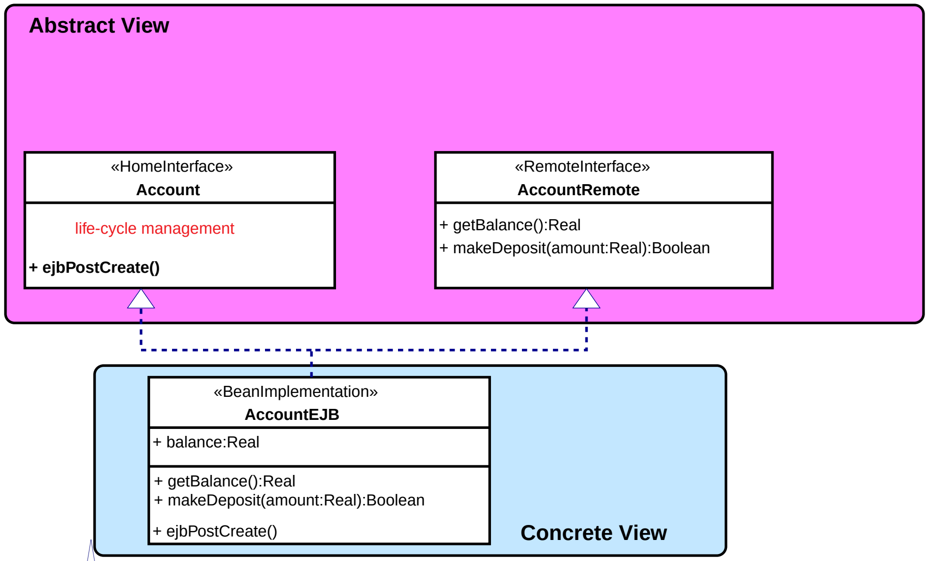
We have chosen the J2EE/EJB Architecture from Sun Microsystems:

- J2EE is an extension of the Java language.
- Provides a wealth of additional services needed for distribution.
- Builds on existing tool support.

Distributed Systems Using J2EE: An EJB

- The “interface” of the EJB is used by the client and the server.
- An EJB is mainly described by three parts:
  - **Home–Interface H:** describing the life–cycle management
  - **Remote–Interface R:** describing the functional behavior
  - **Bean–Implementation I:** implements **H** and **R**
- Special life–cycle management (creation, passivation, deletion).

An example EJB Specification



Abstract View

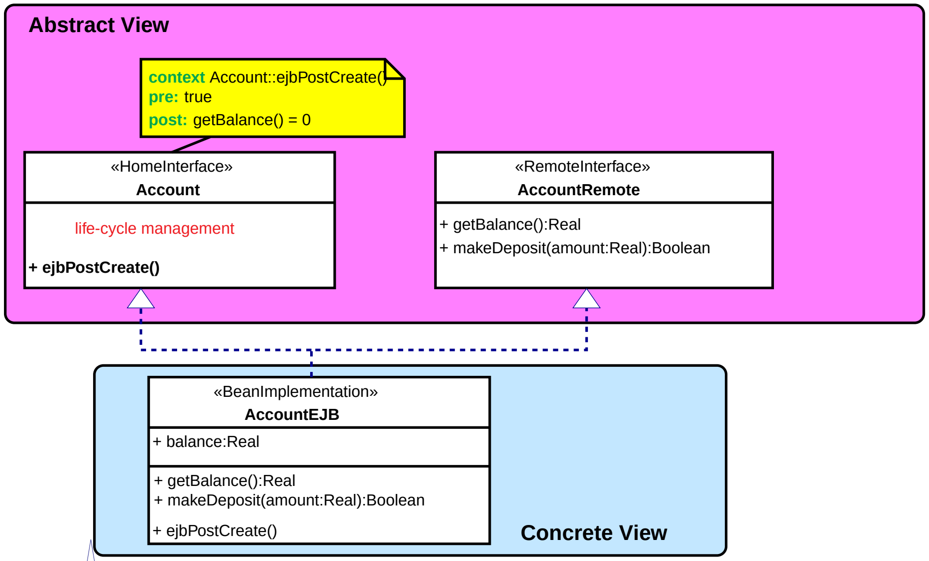
«HomeInterface»
Account
life-cycle management
+ ejbPostCreate()

«RemoteInterface»
AccountRemote
+ getBalance():Real
+ makeDeposit(amount:Real):Boolean

«BeanImplementation»
AccountEJB
+ balance:Real
+ getBalance():Real
+ makeDeposit(amount:Real):Boolean
+ ejbPostCreate()

Concrete View

An example EJB Specification



Abstract View

context Account::ejbPostCreate()
pre: true
post: getBalance() = 0

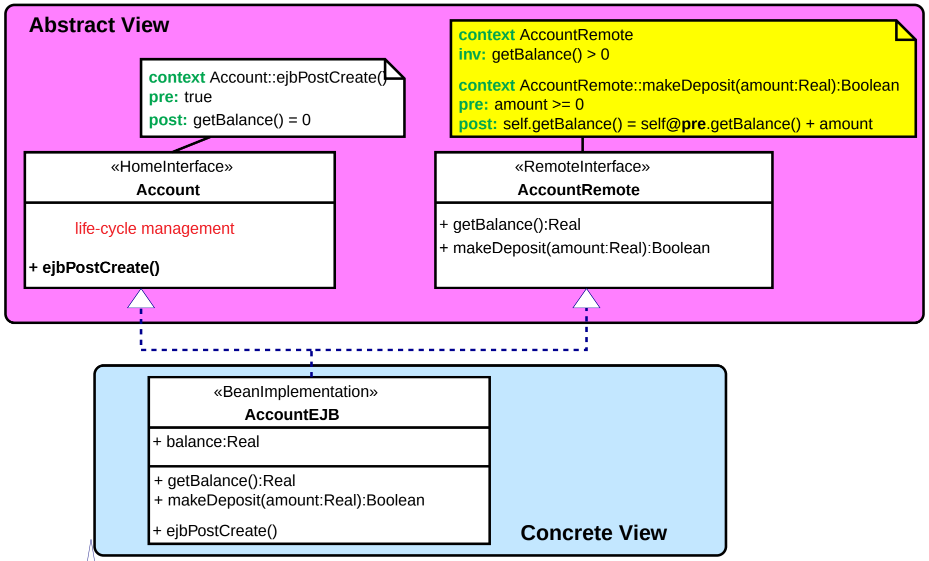
«HomeInterface»
Account
life-cycle management
+ ejbPostCreate()

«RemoteInterface»
AccountRemote
+ getBalance():Real
+ makeDeposit(amount:Real):Boolean

«BeanImplementation»
AccountEJB
+ balance:Real
+ getBalance():Real
+ makeDeposit(amount:Real):Boolean
+ ejbPostCreate()

Concrete View

An example EJB Specification



Abstract View

context Account::ejbPostCreate()
pre: true
post: getBalance() = 0

context AccountRemote
inv: getBalance() > 0

context AccountRemote::makeDeposit(amount:Real):Boolean
pre: amount >= 0
post: self.getBalance() = self@**pre**.getBalance() + amount

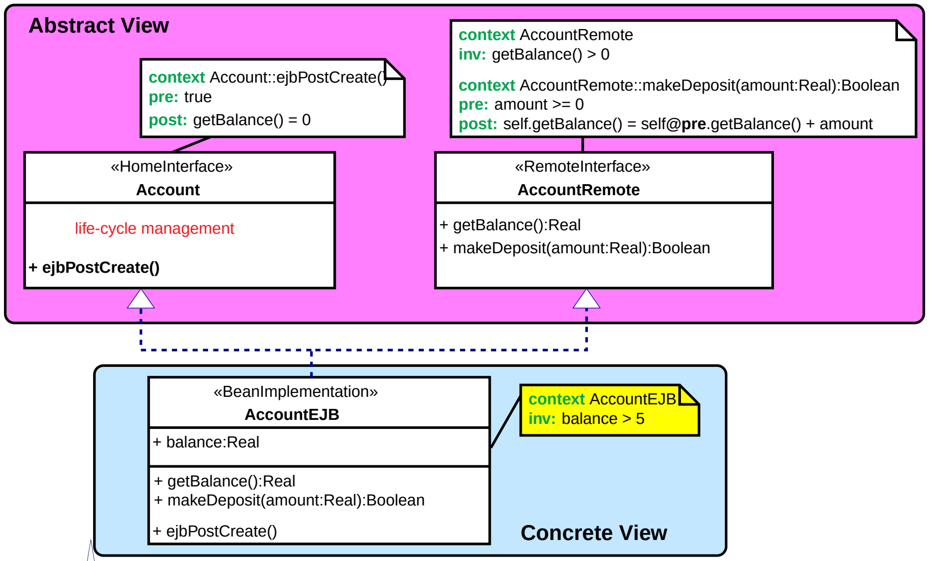
«HomeInterface»
Account
life-cycle management
+ ejbPostCreate()

«RemoteInterface»
AccountRemote
+ getBalance():Real
+ makeDeposit(amount:Real):Boolean

«BeanImplementation»
AccountEJB
+ balance:Real
+ getBalance():Real
+ makeDeposit(amount:Real):Boolean
+ ejbPostCreate()

Concrete View

An example EJB Specification



Abstract View

context Account::ejbPostCreate()
pre: true
post: getBalance() = 0

context AccountRemote
inv: getBalance() > 0

context AccountRemote::makeDeposit(amount:Real):Boolean
pre: amount >= 0
post: self.getBalance() = self@**pre**.getBalance() + amount

«HomeInterface»
Account
life-cycle management
+ ejbPostCreate()

«RemoteInterface»
AccountRemote
+ getBalance():Real
+ makeDeposit(amount:Real):Boolean

«BeanImplementation»
AccountEJB
+ balance:Real
+ getBalance():Real
+ makeDeposit(amount:Real):Boolean
+ ejbPostCreate()

context AccountEJB
inv: balance > 5

Concrete View

Specification of Enterprise Java Beans

J2EE does not provide a concept of a “formal specification” of an EJB.

We fill this gap by adopting OCL. This means:

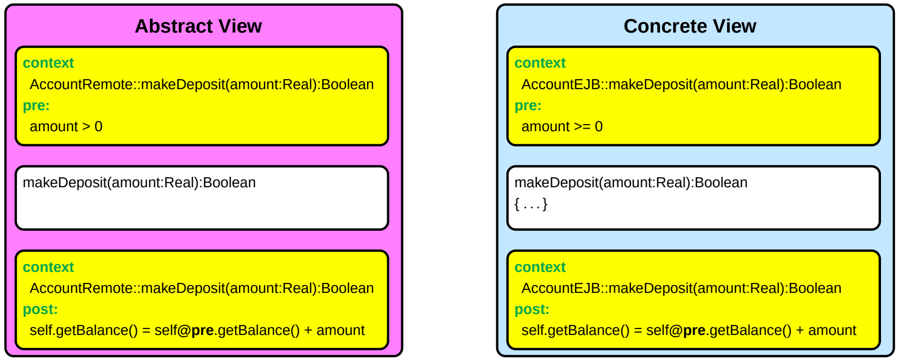
- **Syntactically:**

  Abstract View: The union of the signatures of **H** and **R** extended by further accessor methods for the (public) variables of **I**, and annotated by OCL formulae.

  Concrete View: The bean implementation **I** annotated by OCL formulae.

- **Semantically:**
  What is the operational semantics of OCL formulae written on the different views?

Operational Semantics of the Specification Refinement



Abstract View

context
AccountRemote::makeDeposit(amount:Real):Boolean
pre:
amount > 0

makeDeposit(amount:Real):Boolean

context
AccountRemote::makeDeposit(amount:Real):Boolean
post:
self.getBalance() = self@**pre**.getBalance() + amount

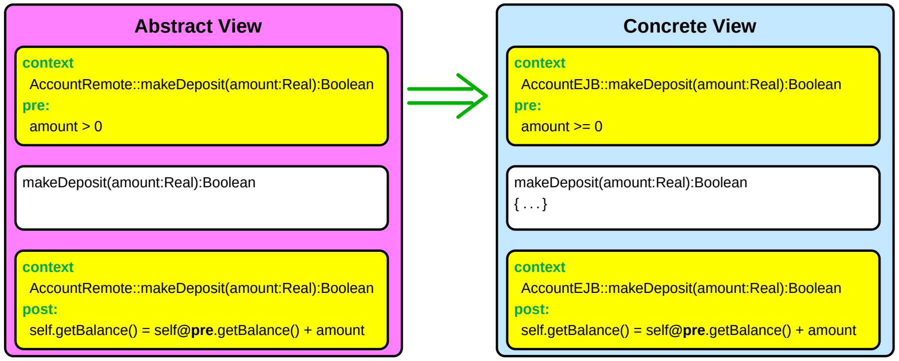
Concrete View

context
AccountEJB::makeDeposit(amount:Real):Boolean
pre:
amount >= 0

makeDeposit(amount:Real):Boolean
{...}

context
AccountEJB::makeDeposit(amount:Real):Boolean
post:
self.getBalance() = self@**pre**.getBalance() + amount

Operational Semantics of the Specification Refinement



Abstract View

context
AccountRemote::makeDeposit(amount:Real):Boolean
pre:
amount > 0

makeDeposit(amount:Real):Boolean

context
AccountRemote::makeDeposit(amount:Real):Boolean
post:
self.getBalance() = self@**pre**.getBalance() + amount

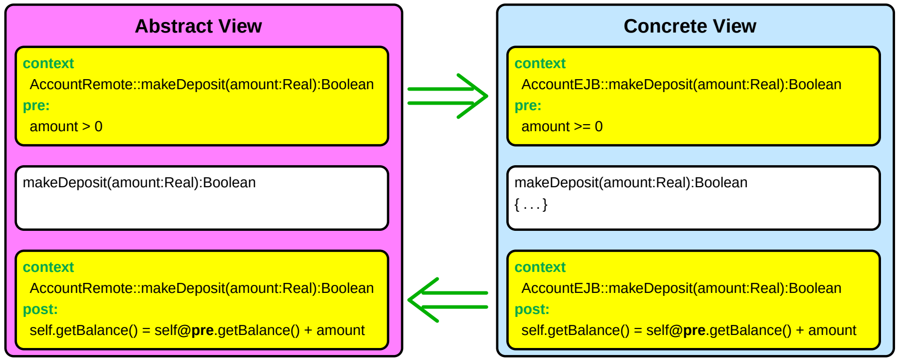
Concrete View

context
AccountEJB::makeDeposit(amount:Real):Boolean
pre:
amount >= 0

makeDeposit(amount:Real):Boolean
{...}

context
AccountEJB::makeDeposit(amount:Real):Boolean
post:
self.getBalance() = self@**pre**.getBalance() + amount

Operational Semantics of the Specification Refinement



Abstract View

context
AccountRemote::makeDeposit(amount:Real):Boolean
pre:
amount > 0

makeDeposit(amount:Real):Boolean

context
AccountRemote::makeDeposit(amount:Real):Boolean
post:
self.getBalance() = self@**pre**.getBalance() + amount

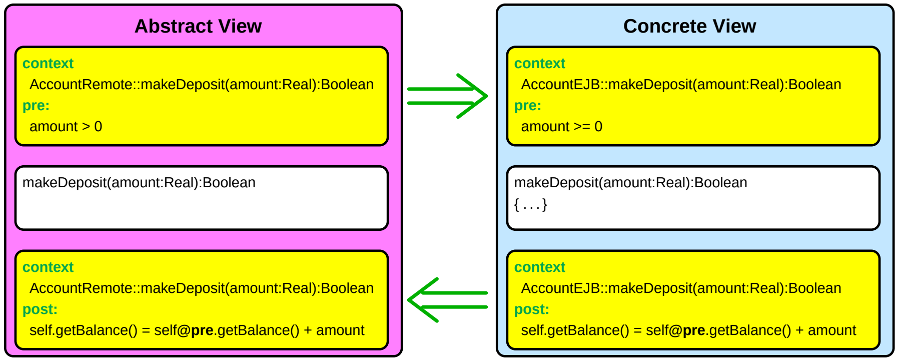
Concrete View

context
AccountEJB::makeDeposit(amount:Real):Boolean
pre:
amount >= 0

makeDeposit(amount:Real):Boolean
{ . . . }

context
AccountEJB::makeDeposit(amount:Real):Boolean
post:
self.getBalance() = self@**pre**.getBalance() + amount

Operational Semantics of the Specification Refinement



Abstract View

context
AccountRemote::makeDeposit(amount:Real):Boolean
pre:
amount > 0

makeDeposit(amount:Real):Boolean

context
AccountRemote::makeDeposit(amount:Real):Boolean
post:
self.getBalance() = self@**pre**.getBalance() + amount

Concrete View

context
AccountEJB::makeDeposit(amount:Real):Boolean
pre:
amount >= 0

makeDeposit(amount:Real):Boolean
{ . . . }

context
AccountEJB::makeDeposit(amount:Real):Boolean
post:
self.getBalance() = self@**pre**.getBalance() + amount

➤ *The Concrete View is a refinement of the Abstract View.*

Operational Semantics of the Specification Black Box Testing

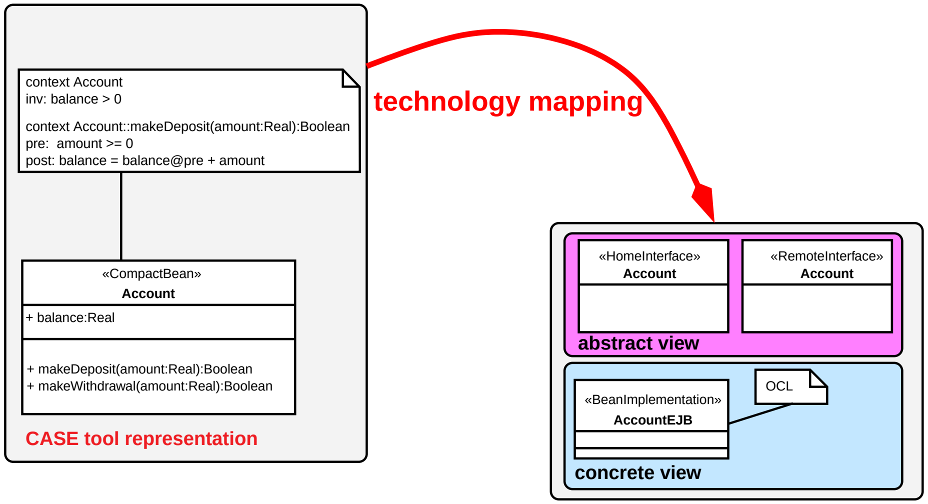
- Based on the specification of the Abstract View:
  - Testing the “external view”.
  - Suited for system implementor using pre–configured components.
- Based on the specification of the Concrete View:
  - Testing the “internal view”.
  - Suited for component developers.

➤ *Runtime OCL constraint checking provides an a posteriori debugging method.*

EJB Design Pattern

- Reduces complexity of specification for test data generation.
- We introduce the concept “**ExpandedBean**” where an EJB can consists out of several home or remote interfaces.
- We provide three patterns:
  - **CompactBean** for standard development: Models an EJB with one remote interface, one home interface and one implementation.
  - **ExpandedBeanHome** for technological optimization:
    Extends the CompactBean by allowing several home interfaces.
  - **ExpandedBeanRemote** for modeling different kind of accesses:
    Extends the CompactBean by allowing several remote interfaces.

The Compact Bean Pattern



```
context Account
inv: balance > 0

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

«CompactBean»
Account

+ balance:Real

+ makeDeposit(amount:Real):Boolean
+ makeWithdrawal(amount:Real):Boolean

CASE tool representation

technology mapping

«HomeInterface»
Account

«RemoteInterface»
Account

abstract view

«BeanImplementation»
AccountEJB

OCL

concrete view

Constraint Checking and Practical Experience

- We integrated a OCL type checker into a CASE Tool for EJB support.
- We integrated a constraint checking code generator into a CASE Tool for EJB support.

- Type checking is based on the tools developed at the University Dresden.
- Runtime checking is done via “method–wise” wrapping code.
- Internal method invocations are checked.

Further Work

- Specification of transactions.

- Systematic generation of test–data based on the OCL specification.

- Formal analysis of the relation between Abstract View and Concrete View.

➤ *We will develop a declarative semantics of OCL, which is done through an embedding of OCL into an theorem prover.*

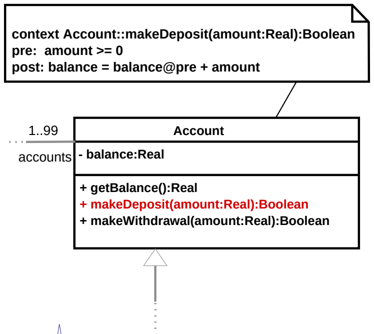
Appendix

Back

UML and OCL

UML

- several diagram types
- diagrammatic method
- metamodeling approach
- OMG standard
- widely accepted (in industry)



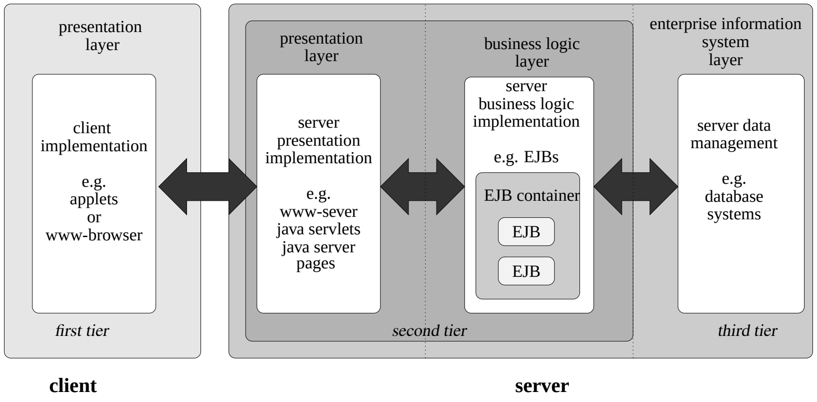
```
context Account::makeDeposit(amount:Real):Boolean
pre:  amount >= 0
post: balance = balance@pre + amount
```

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

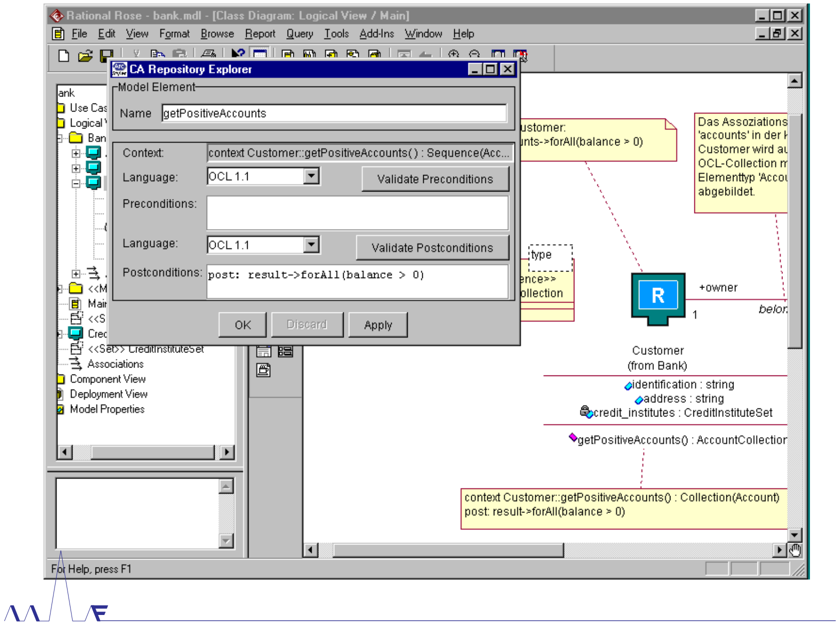
OCL

- textual extension
- based on logic and set theory
- designed for annotation of UML diagrams
- class–diagrams:
  - preconditions
  - postconditions
  - invariants
- part of the UML
- no declarative semantics

The J2EE Application Model



Modeling EJBs using UML/OCL with ArcStyler



Rational Rose - bank.mdl - [Class Diagram: Logical View / Main]

CA Repository Explorer

Model Element

Name: getPositiveAccounts

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

Language: OCL 1.1

Preconditions:

Language: OCL 1.1

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

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

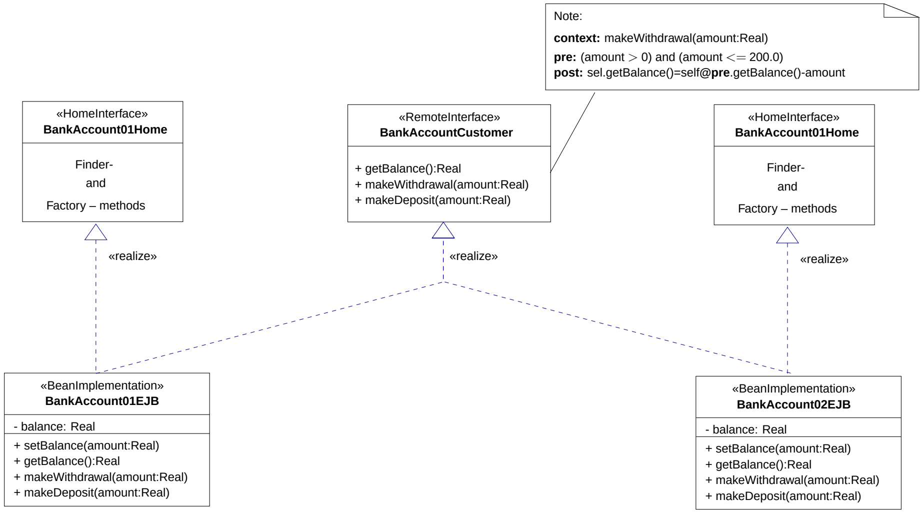
context Customer::getPositiveAccounts() : Collection(Account)
post: result->forAll(balance > 0)

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

ustomer:
nts->forAll(balance > 0)

For Help, press F1

The ExpandedBeanHome Pattern

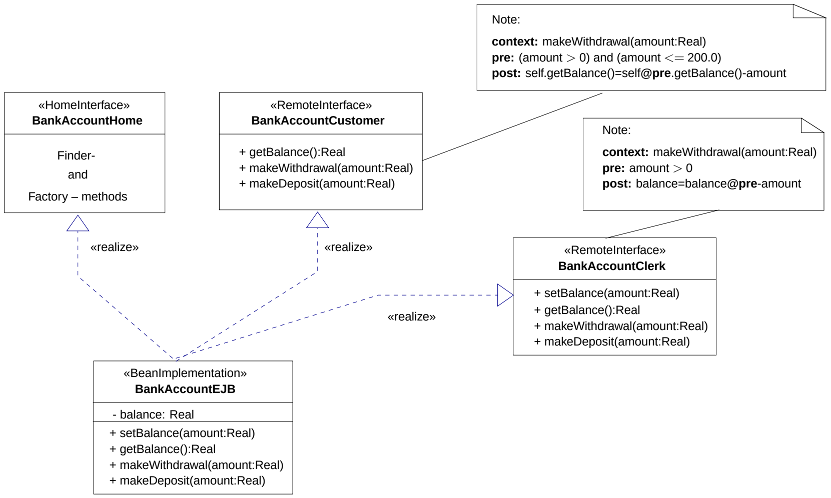


Note:

context: makeWithdrawal(amount:Real)

pre: (amount $>$ 0) and (amount $<=$ 200.0)
post: sel.getBalance()=self@**pre**.getBalance()-amount

The ExpandedBeanRemote Pattern



Note:

context: makeWithdrawal(amount:Real)
pre: (amount $>$ 0) and (amount $<=$ 200.0)
post: self.getBalance()=self@**pre**.getBalance()-amount

Note:

context: makeWithdrawal(amount:Real)
pre: amount $>$ 0
post: balance=balance@**pre**-amount

«HomeInterface»
BankAccountHome

Finder-
and
Factory – methods

«RemoteInterface»
BankAccountCustomer

+ getBalance():Real
+ makeWithdrawal(amount:Real)
+ makeDeposit(amount:Real)

«RemoteInterface»
BankAccountClerk

+ setBalance(amount:Real)
+ getBalance():Real
+ makeWithdrawal(amount:Real)
+ makeDeposit(amount:Real)

«realize»

«realize»

«realize»

«BeanImplementation»
BankAccountEJB

- balance: Real

+ setBalance(amount:Real)
+ getBalance():Real
+ makeWithdrawal(amount:Real)
+ makeDeposit(amount:Real)