TU Clausthal

SSE

Modelling CoCoME with DisCComp

Dagstuhl Workshop, 02.08.2007

Sebastian Herold

Clausthal University of Technology
Department of Informatics – Software Systems Engineering
Chair of Prof. Dr. Andreas Rausch

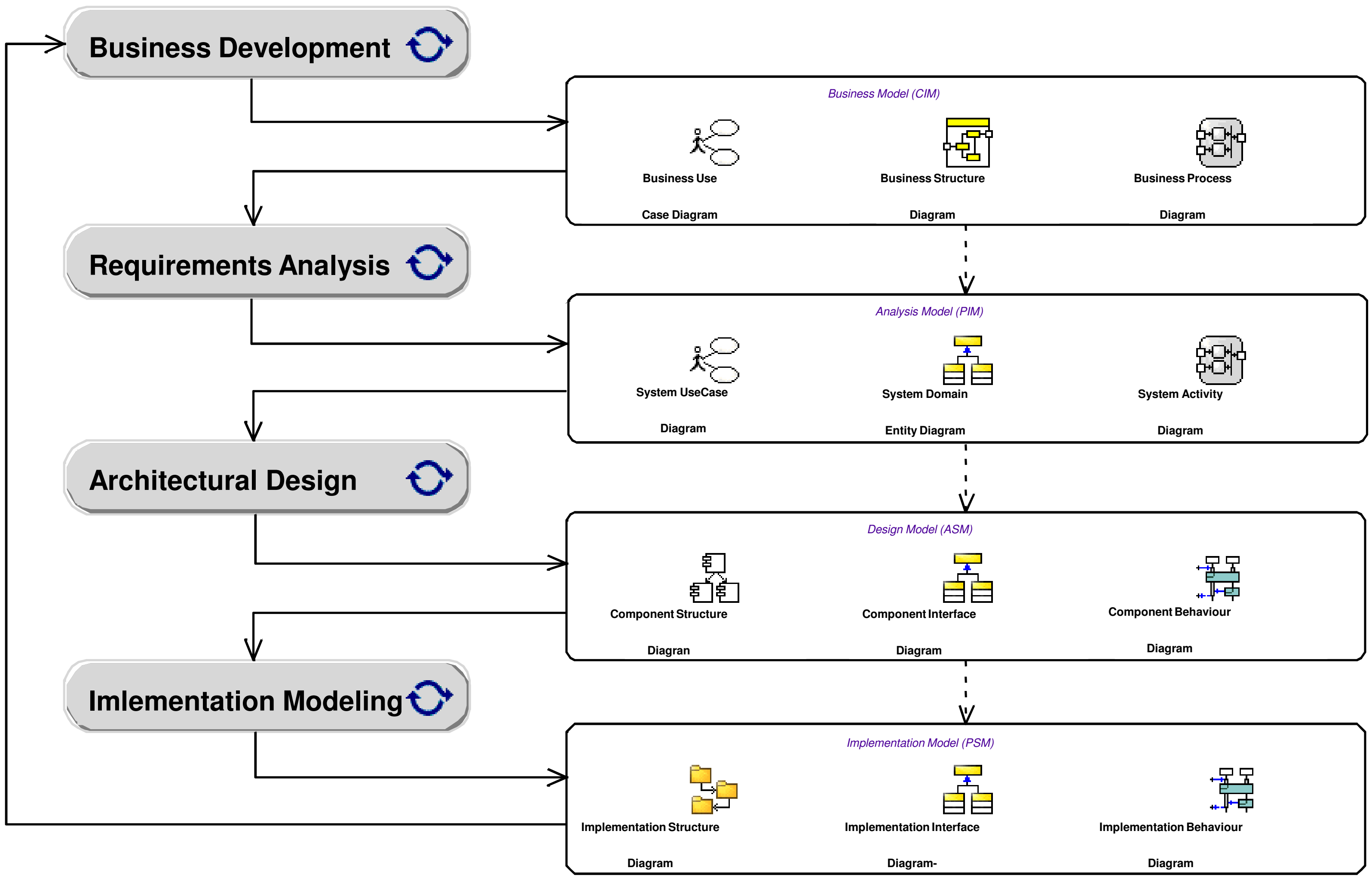
Overview

- Introduction
  - The team
- The DisCComp Approach
  - History of DisCComp and motivation for participating in the contest
  - Foundations of the system model (formal semantics)
  - Foundations of the specification technique
- The Modelled CoCoME Cutout
  - Static view
  - Behavioural view
- Conclusion
  - Experiences, limitations
  - Future work

The DisCComp Team

- **Affiliation**
  - TU Clausthal, Software Systems Engineering Group (formerly known as Software Architecture Group from Kaiserslautern)
- **Members**
  - André Appel, Holger Klus, Andreas Rausch, Sebastian Herold
- **Component Approach**
  - DisCComp: A Formal Model for Distributed Concurrent Components
- **Specification Technique**
  - UML-based, OCL-based
- **Experiences**
  - Seamless UML software/system modeling
  - Software architecture in general

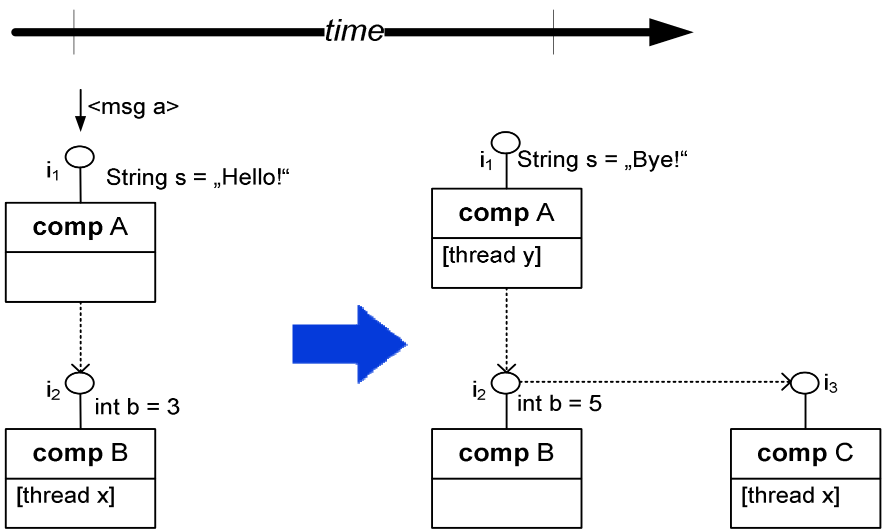
Scope and Purpose of DisCComp



The DisCComp Approach

DisCComp: Introduction

- DisCComp: set-theoretic formalization of distributed concurrent components which allows
  - synchronous and asynchronous messages
  - a shared global state
  - dynamically changing structures



DisCComp: Introduction

- History
  - Early versions only supported asynchronous communication
  - Early versions based on timed streams (global clock)
  - Specification technique extended UML 1.x (due to missing features of UML)
  - Specification technique was not extended according to system model extensions
  - Specification technique partly hard to use (contracts)
- Motivation for participating in the contest
  - New specification technique required:
    - based on UML2 if possible
    - with better usability
    - consistent to the formal system model
  - Answering the question: Can systems of practically relevant size and functionality be modelled with DisCComp?

- Instances in a system s:

$$\mathrm{Instance}_s := \mathrm{System}_s \cup \mathrm{Component}_s \cup \mathrm{Interface}_s \cup \mathrm{Attribute}_s \cup \mathrm{Connection}_s \cup \mathrm{Message}_s \cup \mathrm{Call}_s \cup \mathrm{Thread}_s \cup \mathrm{Value}_s$$

- The system state
  - Structural state

$$\mathrm{alive}_s := \mathrm{Instance}_s \to \mathrm{BOOLEAN}$$
$$\mathrm{assignment}_s := \mathrm{Interface}_s \to \mathrm{Component}_s$$
$$\mathrm{allocation}_s := \mathrm{Attribute}_s \to \mathrm{Interface}_s$$
$$\mathrm{connects}_s := \mathrm{Connection}_s \to \{(\mathrm{from}, \mathrm{to}) \mid \mathrm{from} \in \mathrm{Component}_s \cup \mathrm{Interface}_s, \mathrm{to} \in \mathrm{Interface}_s\}$$

  - Valuation state

$$\mathrm{valuation}_s := \mathrm{Attribute}_s \to \mathrm{Value}_s$$

DisCComp: The System Model

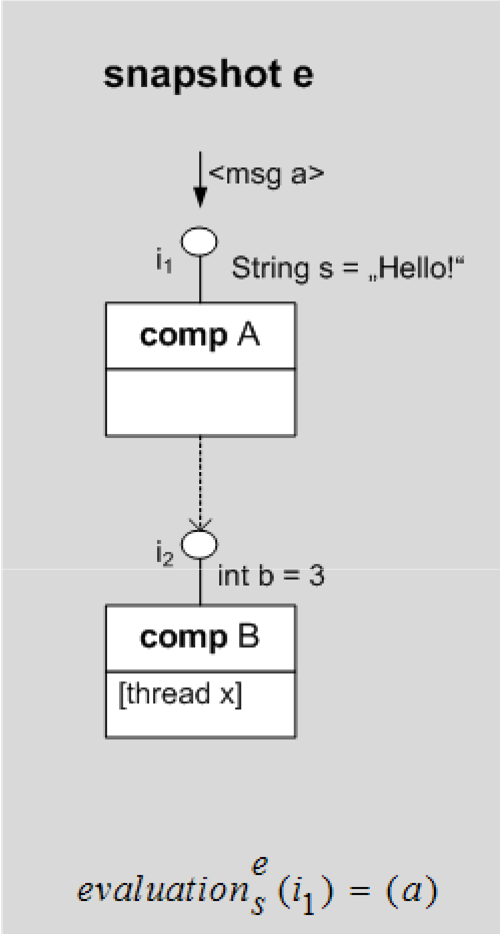
- The system state
  - Communication state

    $$\mathrm{evaluation}_s : \mathrm{Interface}_s \to \mathrm{Message}_s^*$$

  - Execution state

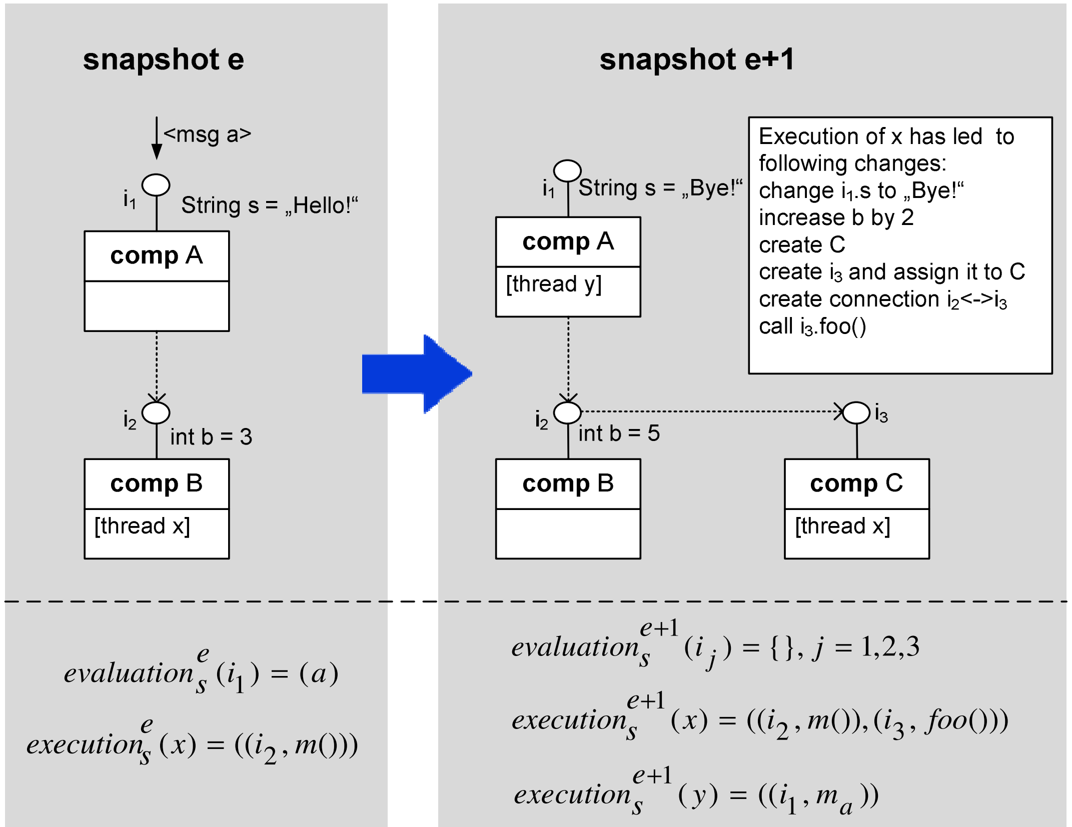
    $$\mathrm{execution}_s : \mathrm{Thread}_s \to (\mathrm{Interface}_s \times \mathrm{Call}_s)^*$$

- The system's overall state at some point in time e is denoted as *snapshot*:



$$\mathrm{snapshot}_s^e : alive_s^e \times assignment_s^e \times allocation_s^e \times connects_s^e \times valuation_s^e \times evaluation_s^e \times execution_s^e$$

Thread Behaviour: What Happens Between Snapshots?



- A thread is selected for execution (runtime environment).
- Pending asynchronous messages are processed, threads are created.
- Changes, the threads requires, are computed by:

$$\text{behaviour}_s : \text{Thread}_s \times \text{Snapshot}_s \rightarrow \text{Snapshot}_s$$

System Behaviour: Composition of Thread Behaviours

- Operator to replace elements in sets (relations):

$$X \lhd Y := \{ a \mid a \in Y \vee (a \in X \wedge \pi_1(\{a\}) \cap \pi_1(Y) = \{\}) \}$$

- Composing the system behaviour (=computing the next snapshot)

$next\_snapshot(snapshot_s^e) := snapshot_s^{e+1} = (alive_s^{e+1}, assignment_s^{e+1}, ...)$ with

$alive_s^{e+1} = alive_s^e \lhd (\pi_1(behaviour(snapshot_s^e, next\_thread())) \lhd \pi_1(message\_execution(snapshot_s^e)))$

$assignment_s^{e+1} = assignment_s^e \lhd \pi_2(behaviour(snapshot_s^e, next\_thread()))$

$allocation_s^{e+1} = allocation_s^e \lhd \pi_3(behaviour(snapshot_s^e, next\_thread()))$

$connects_s^{e+1} = connects_s^e \lhd \pi_4(behaviour(snapshot_s^e, next\_thread()))$

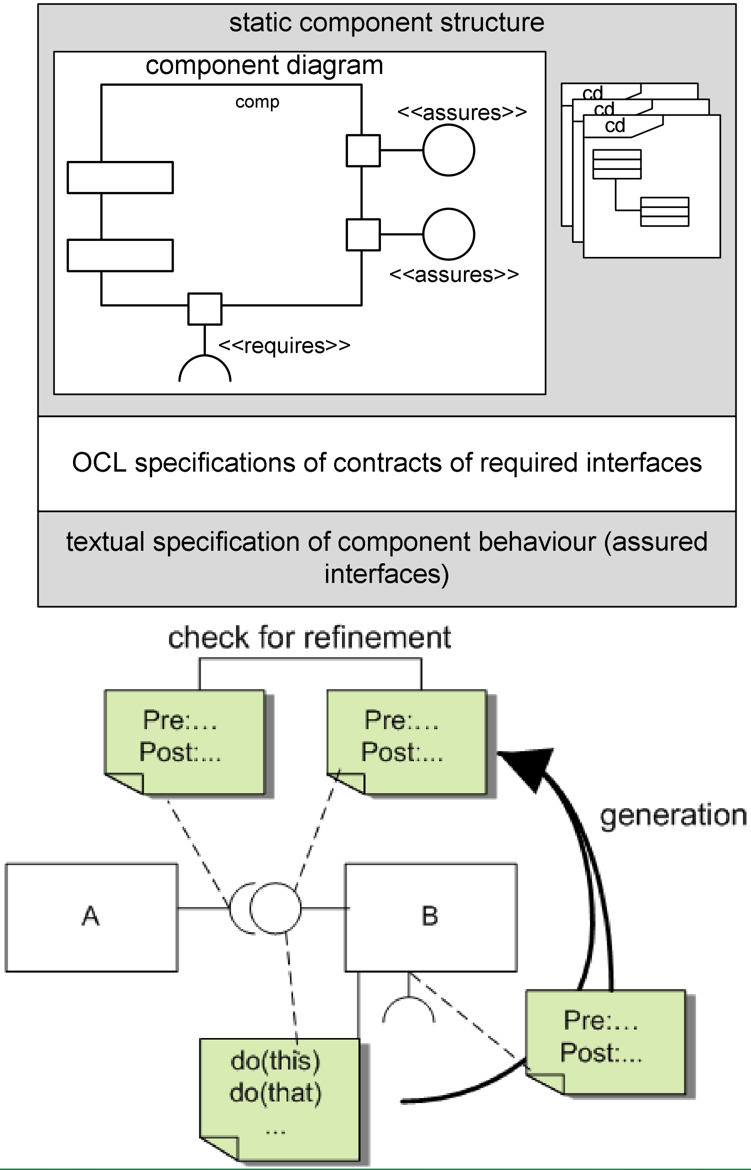
$valuation_s^{e+1} = valuation_s^e \lhd \pi_5(behaviour(snapshot_s^e, next\_thread()))$

$evaluation_s^{e+1} = evaluation_s^e \lhd \pi_6(behaviour(snapshot_s^e, next\_thread()))$

$execution_s^{e+1} = execution_s^e \lhd$

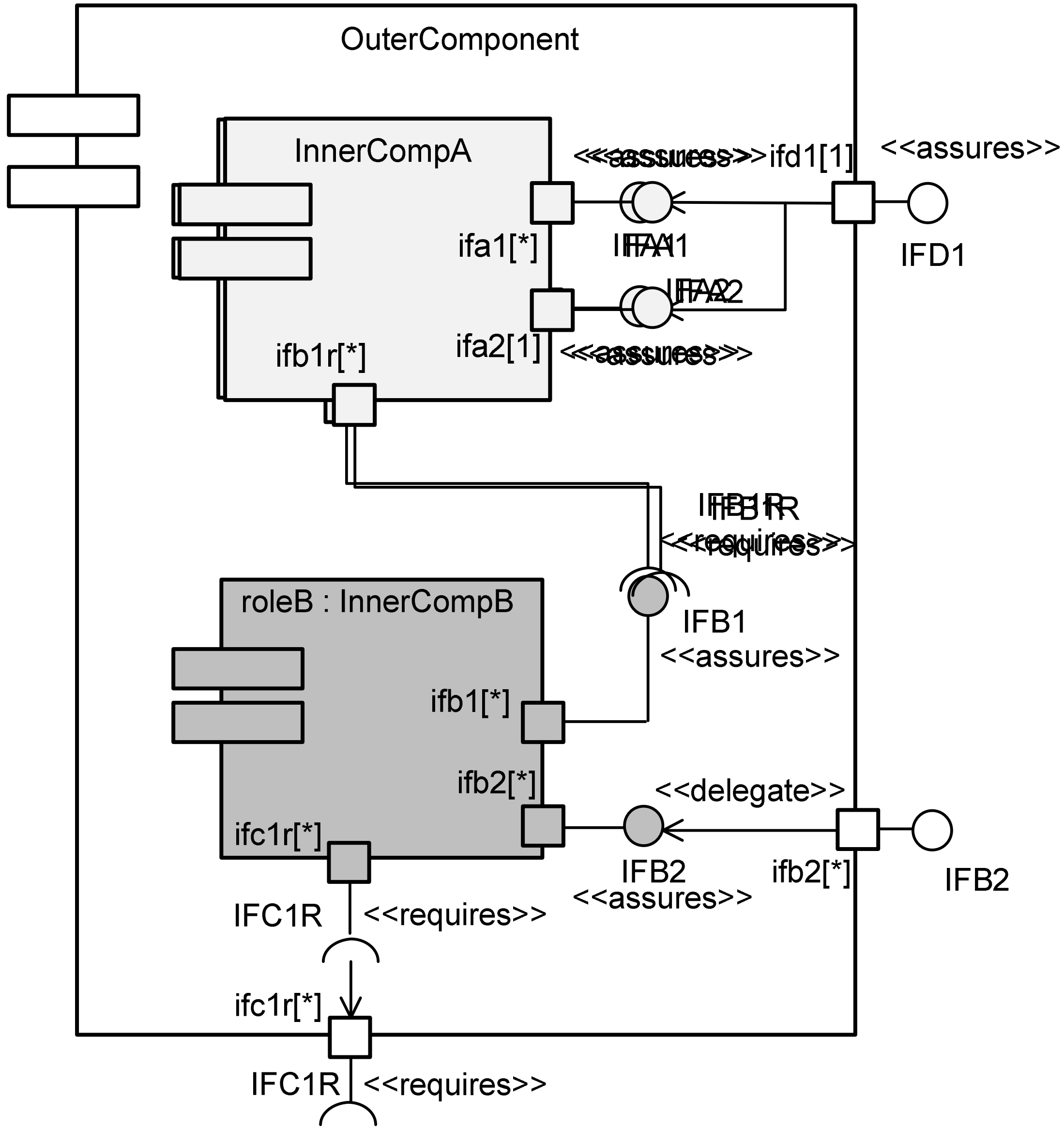
$(\pi_7(behaviour(snapshot_s^e, next\_thread())) \lhd \pi_7(message\_execution(snapshot_s^e)))$

DisCComp Specifications



- State of the DisCComp specification art
  - Remember: current state of specification technique does not reflect the state of system model (synchronous method calls)
  - UML 1.x -> UML 2.1
  - specification of pre-/post-conditions causes massive overhead
- Main idea:
  - Static description: UML component and class diagrams
  - Abstract behaviour description of required interfaces by using OCL invariants, pre- and post-conditions
  - Textual (imperative) behaviour specification of assured interfaces
  - Generation of pre- and post-conditions for assured interfaces by analyzing imperative specifications, when wiring components

Static View: Component Diagrams



Textual Specification of Interface Behaviour

- Focus here: extend existing fine-grained language by introducing some keywords with defined semantics.
- For example, creating instances:
  - `ifInst : IfType =` **`NEW INTERFACE`** `IfType [`**`CONNECT BY`** `ConnType]`

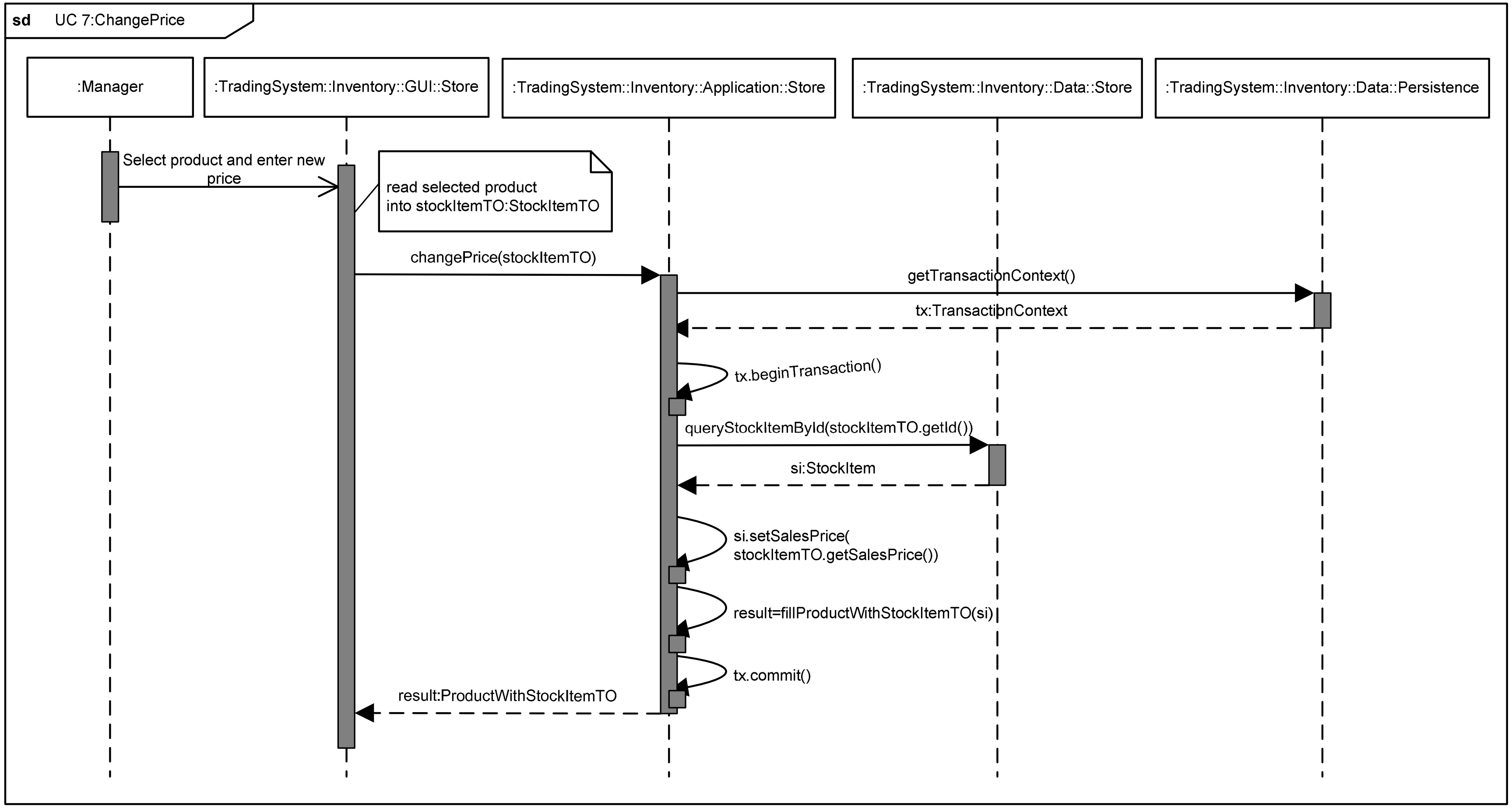
    Create new interface instance of type `IfType`. Assign it to the "current component". Connect it with current interface (optional).
  - `connInst : ConnType =` **`NEW CONNECTION`** `ConnType TO ifInst`

    Create new connection between the current interface and `ifInst`. Types must be consistent to the component and class diagrams.
- Return values:
  - **`CONNECT`** `ifInst` **`TO CALLER AND REASSIGN`**

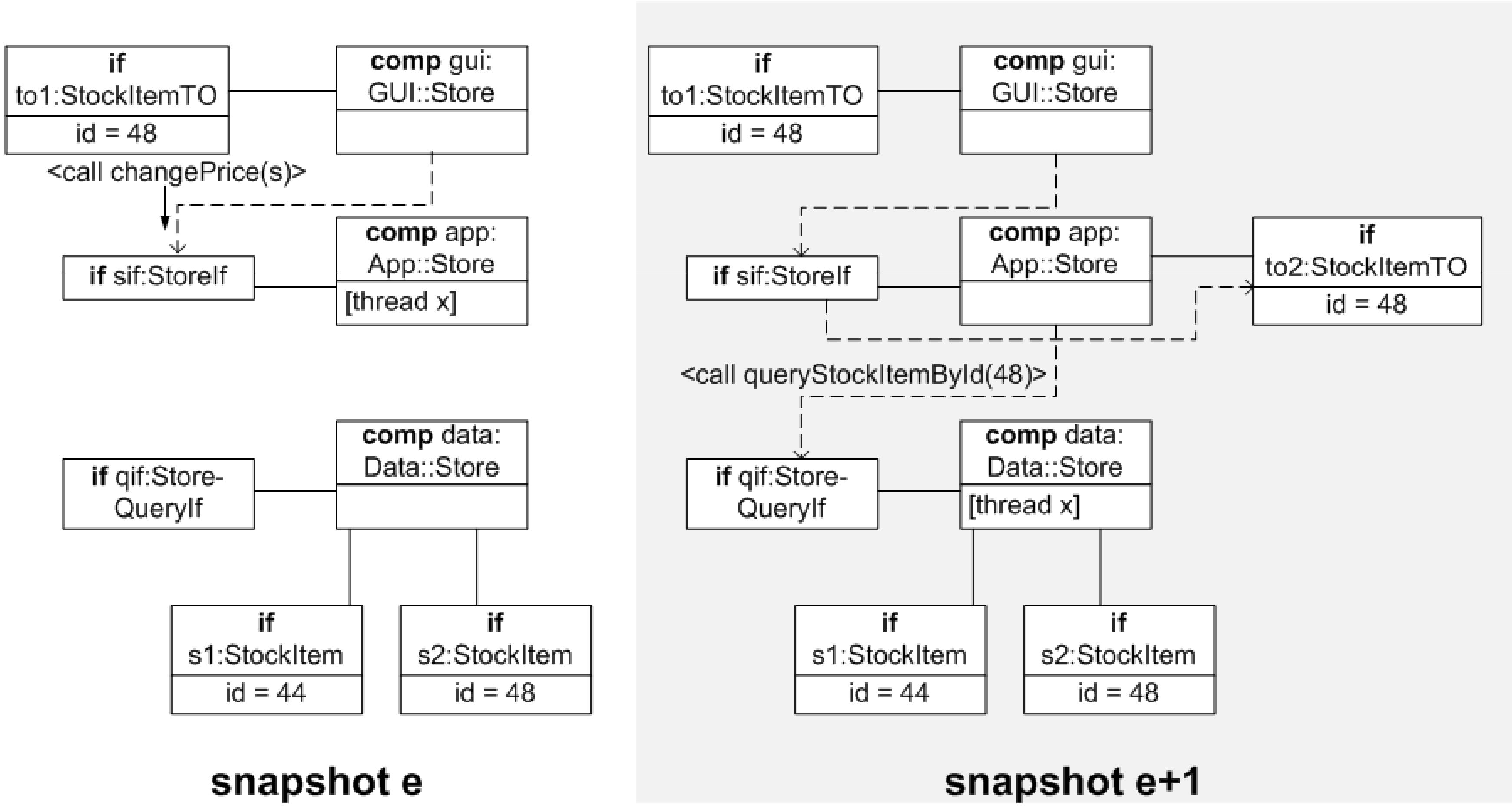
    Leave method, return to calling interface, and leave `ifInst` to the calling component.

The Modelled CoCoME Cutout

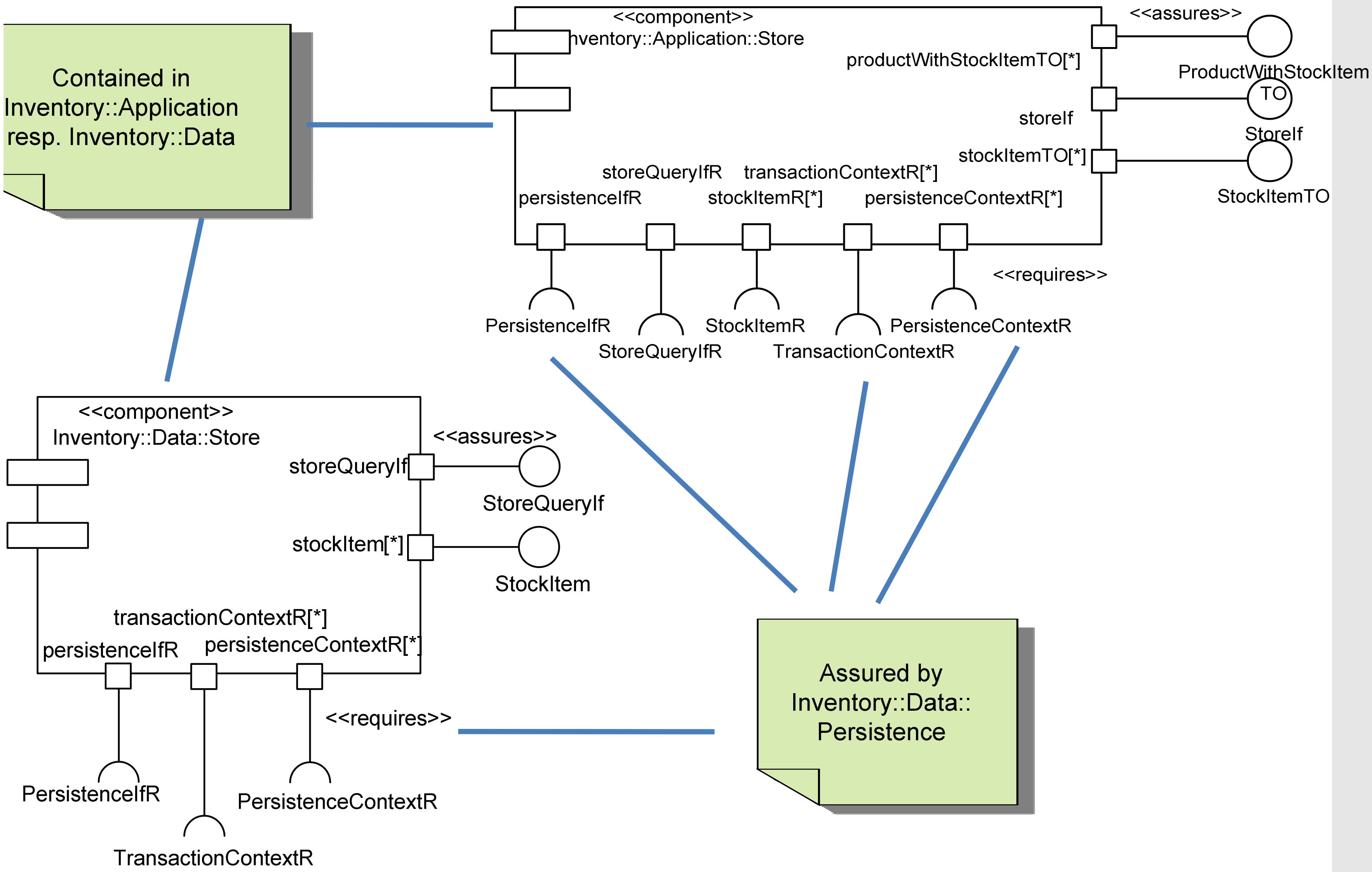
Modelled Cutout: Use Case *ChangePrice*



- Simplified behaviour (without technical components)



Static Specification: Atomic Components



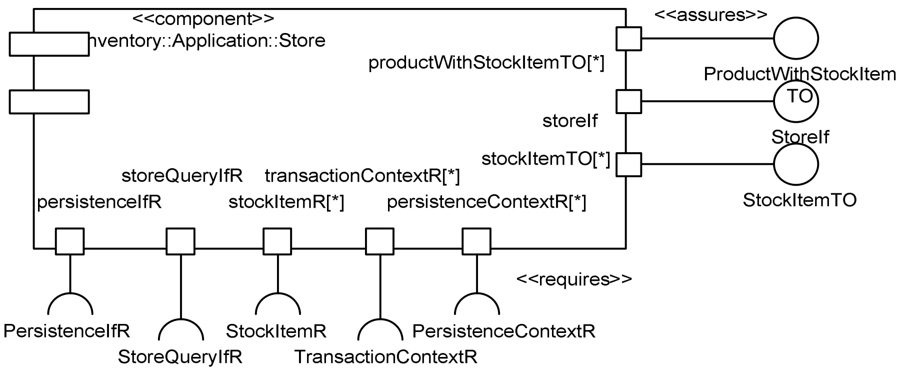
Textual Description of Required Interfaces

```
INTERFACE StoreQueryIfR
    METHOD queryStockItemById(long sId): StockItemR
     Pre: sId >= 0
     Post: let queriedItems : Set(StockItemR) = stockItemR->select(s|s.getId()=sId) in
      if queriedItems->notEmpty then
            result = queriedItems->first();
         else
            result = NULL
         endif
    END METHOD
  END INTERFACE
```

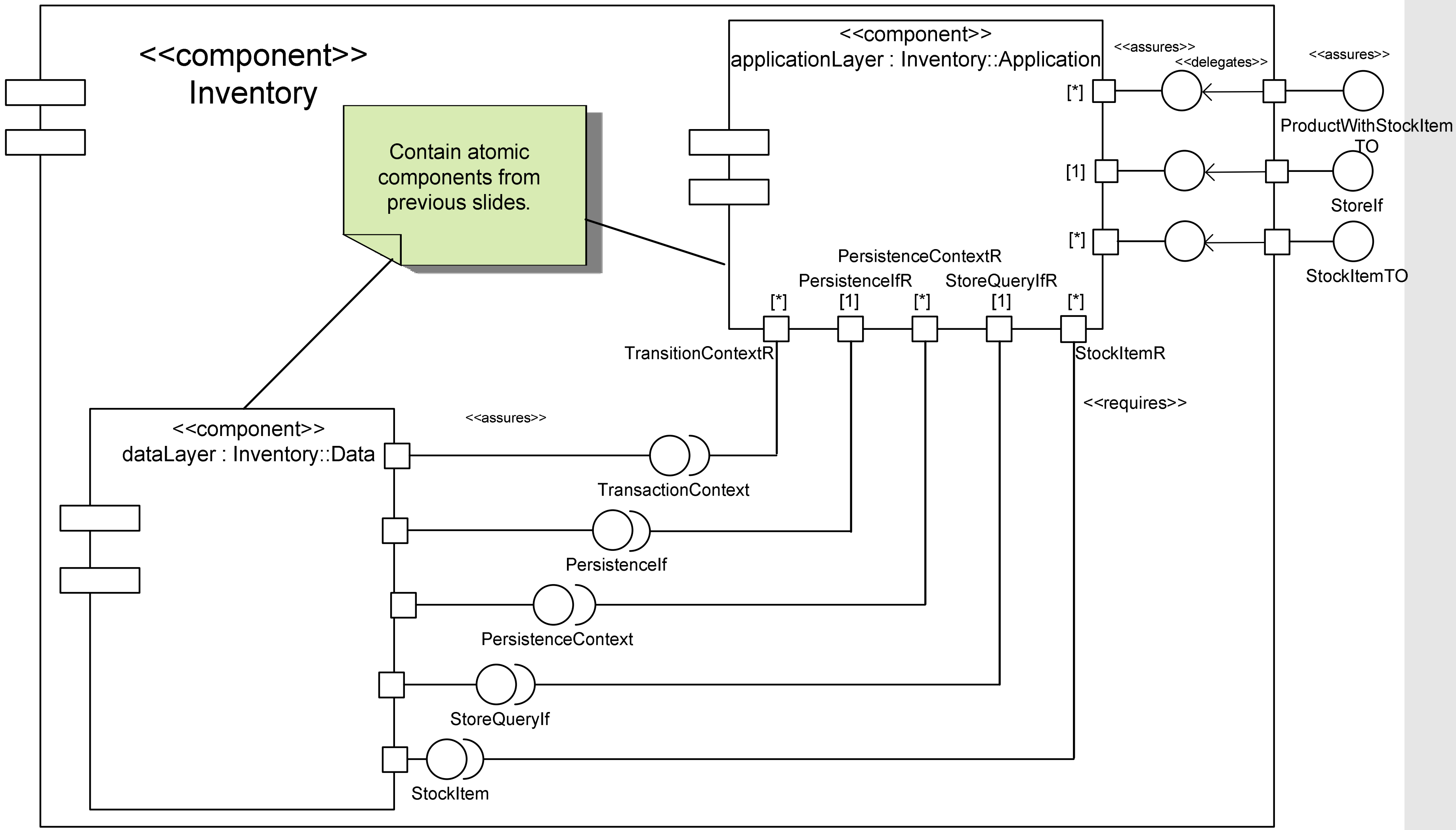
```
INTERFACE StockItemR
    METHOD getId():long
     Post: result = self@pre.getId()
    END METHOD
    ...
END INTERFACE
```

Textual description of Assured Interfaces

```
INTERFACE StoreIf
METHOD changePrice(StockItemTO stockItemTO) : ProductWithStockItemTO
      result : ProductWithStockItemTO:=NEW INTERFACE ProductWithStockItemTO;
      pctx : PersistenceContextR:=persistenceIfR.getPersistenceContext();
      tx : TransactionContextR :=pif.getTransactionContext();
      tx.beginTransaction();
      si : StockItemR := storequeryIfR.queryStockItemById(stockItemTO.getId());
      IF (si != NULL) THEN
                 .
                 .            //copy data to result object if si != NULL
                 .
         CONNECT result TO CALLER AND REASSIGN;
      ENDIF
      RETURN NULL;
END METHOD
```



Hierarchical Components



Conclusion

Summary and Experiences

- DisCComp provides a formal model for distributed concurrent components
  - Supports asynchronous and synchronous communication (as required in CoCoME)
  - Specification technique partly based on UML and OCL, modular specifications by contracts
- Lessons learned
  - Adequate specification technique: we modelled the cutout rather quickly (compared to early DisCComp specifications)
  - We were able to model the functionality of the cutout in terms of DisCComp
  - OCL is troublesome

Limitations and Future Work

- Limitations
  - Non-functional properties are not considered
- Future work
  - Semantic foundation of specification technique has to be completed
  - Generation of pre- and post-conditions: what is possible?
  - Extend tool support
    - Specification tool *DesignIt* has to be modified according to new specification technique
    - Extension for generation as mentioned above

Thank you for your attention!

Any Questions