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

Introduction

The DisCComp Approach

The Modelled CoCoME Cutout

Conclusion

- Business Development
  - Business Model (CIM)
  - Analysis Model (PIM)
  - Business Development
  - Requirements Analysis
  - System Activity Diagram
  - System Domain Entity Diagram
  - System Use Case Diagram
  - Analysis Model (PIM)
  - Design Model (ASM)
  - Implementation Model (PSM)
  - Architectural Design
  - Implementation Modeling
  - Component Structure Diagram
  - Component Interface Diagram
  - Component Behaviour Diagram
  - Implementation Structure Diagram
  - Implementation Interface Diagram
  - Implementation Behaviour Diagram

Modelling CoCoME with DisCComp
©Sebastian Herold

02.08.2007
The DisCComp Approach
• DisCComp: set-theoretic formalization of distributed concurrent components which allows
  – synchronous and asynchronous messages
  – a shared global state
  – dynamically changing structures
• **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:**

\[
\text{Instance}_s := \text{System}_s \cup \text{Component}_s \cup \text{Interface}_s \cup \text{Attribute}_s \cup \text{Connection}_s \cup \\
\text{Message}_s \cup \text{Call}_s \cup \text{Thread}_s \cup \text{Value}_s
\]

• **The system state**
  
  – **Structural state**

\[
\text{alive}_s := \text{Instance}_s \rightarrow \text{BOOLEAN}
\]

\[
\text{assignment}_s := \text{Interface}_s \rightarrow \text{Component}_s
\]

\[
\text{allocation}_s := \text{Attribute}_s \rightarrow \text{Interface}_s
\]

\[
\text{connects}_s := \text{Connection}_s \rightarrow \{(\text{from}, \text{to}) \mid \text{from} \in \text{Component}_s \cup \text{Interface}_s, \text{to} \in \text{Interface}_s\}
\]

  – **Valuation state**

\[
\text{valuation}_s := \text{Attribute}_s \rightarrow \text{Value}_s
\]
• The system state
  – Communication state
    \( \text{evaluation}_s : \text{Interface}_s \rightarrow \text{Message}_s^* \)
  – Execution state
    \( \text{execution}_s : \text{Thread}_s \rightarrow (\text{Interface}_s \times \text{Call}_s)^* \)

• The system’s overall state at some point in time \( e \) is denoted as \( \text{snapshot} \):

\[
\text{snapshot}_s^e : \text{alive}_s^e \times \text{assignment}_s^e \times \text{allocation}_s^e \times \text{connects}_s^e \times \\
\text{valuation}_s^e \times \text{evaluation}_s^e \times \text{execution}_s^e
\]
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:

\[
\text{behaviour}_s : \text{Thread}_s \times \text{Snapshot}_s \rightarrow \text{Snapshot}_s
\]
• Operator to replace elements in sets (relations):

\[ X \triangleleft Y := \{ a \mid a \in Y \lor (a \in X \land \pi_1(\{a\}) \cap \pi_1(Y) = \{\})\} \]

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

\[ \text{next\_snapshot}(\text{snapshot}^e_s) := \text{snapshot}^{e+1}_s = (\text{alive}^{e+1}_s, \text{assignment}^{e+1}_s,...) \text{ with} \]

\[ \text{alive}^{e+1}_s = \text{alive}^e_s \triangleleft (\pi_1(\text{behaviour}(\text{snapshot}^e_s, \text{next\_thread}()))) \triangleleft \pi_1(\text{message\_execution}(\text{snapshot}^e_s))) \]

\[ \text{assignment}^{e+1}_s = \text{assignment}^e_s \triangleleft \pi_2(\text{behaviour}(\text{snapshot}^e_s, \text{next\_thread}())) \]

\[ \text{allocation}^{e+1}_s = \text{allocation}^e_s \triangleleft \pi_3(\text{behaviour}(\text{snapshot}^e_s, \text{next\_thread}())) \]

\[ \text{connects}^{e+1}_s = \text{connects}^e_s \triangleleft \pi_4(\text{behaviour}(\text{snapshot}^e_s, \text{next\_thread}())) \]

\[ \text{valuation}^{e+1}_s = \text{valuation}^e_s \triangleleft \pi_5(\text{behaviour}(\text{snapshot}^e_s, \text{next\_thread}())) \]

\[ \text{evaluation}^{e+1}_s = \text{evaluation}^e_s \triangleleft \pi_6(\text{behaviour}(\text{snapshot}^e_s, \text{next\_thread}())) \]

\[ \text{execution}^{e+1}_s = \text{execution}^e_s \triangleleft \]

\[ (\pi_7(\text{behaviour}(\text{snapshot}^e_s, \text{next\_thread}()))) \triangleleft \pi_7(\text{message\_execution}(\text{snapshot}^e_s))) \]
The DisCComp Approach

- **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
• 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
• Simplified behaviour (without technical components)

**Snapshot e**
- `if to1:StockItemTO
  id = 48
  <call changePrice(s)>`
- `if sif:StoreIf`
- `if qif:Store-QueryIf`
- `if s1:StockItem
  id = 44`
- `if s2:StockItem
  id = 48`

**Snapshot e+1**
- `if to2:StockItemTO
  id = 48
  <call queryStockItemByld(48)>`
- `if sif:StoreIf`
Static Specification: Atomic Components

- Contained in Inventory::Application resp. Inventory::Data
- <component>
  - Inventory::Application::Store
    - storeQueryIf
    - stockItem[*]
    - transactionContextR[*]
    - persistenceIfR
    - persistenceContextR[*]
    - <<requires>>
  - transactionContextR[*]
  - persistenceContextR[*]
  - <<requires>>
- <<assures>>
  - ProductWithStockItem
    - StoreIf
    - stockItemTO[*]
    - persistenceContextR[*]
    - <<requires>>
    - Store
- Assured by Inventory::Data::Persistence
INTERFACE StoreQueryIfR
  METHOD queryStockItemById(long sId): StockItemR
    Pre: sId >= 0
    Post: let queriedItems : Set(StockItemR) = stockItemR->select(s | s.getItemId() = 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.getItemId()
  END METHOD
...
END INTERFACE
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.queryStockItemByld(stockItemTO.getld());
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

<<component>>
Inventory

Contains atomic components from previous slides.

<<component>>
dataLayer : Inventory : Data

<<assures>>
TransactionContext

<<assures>>
Persistencelf

<<assures>>
PersistenceContext

<<requires>>
StockItem

<<delegates>>
PersistencelfR

<<assures>>
StoreQueryIf

<<assures>>
PersistencelfR

<<requires>>
StockItemR

<<assures>>
ProductWithStockItem

<<assures>>
StoreIf

<<assures>>
StockItemTO

<<assures>>
TransitionContextR

<<assures>>
StoreQueryR

<<assures>>
StockItemR

<<assures>>
ProductWithStockItem
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
  – 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