

Harnessing Theories for Tool Support in Software

The CoCoME experience

Volker Stolz

vs@iist.unu.edu



United Nations
University

UNU-IIST

International Institute for
Software Technology



Using rCOS in Anger

CoCoME (Common Component Modelling Example):

- common problem description
- component-based
- various aspects to formally model and analyse
- generate code

Case study:

- Point of Sale Terminal: cashdesk with GUI and peripherals
- connected to Store-server using a message bus
- Enterprise-server connected to various stores via RMI
- Use cases of (inter)actions between entities

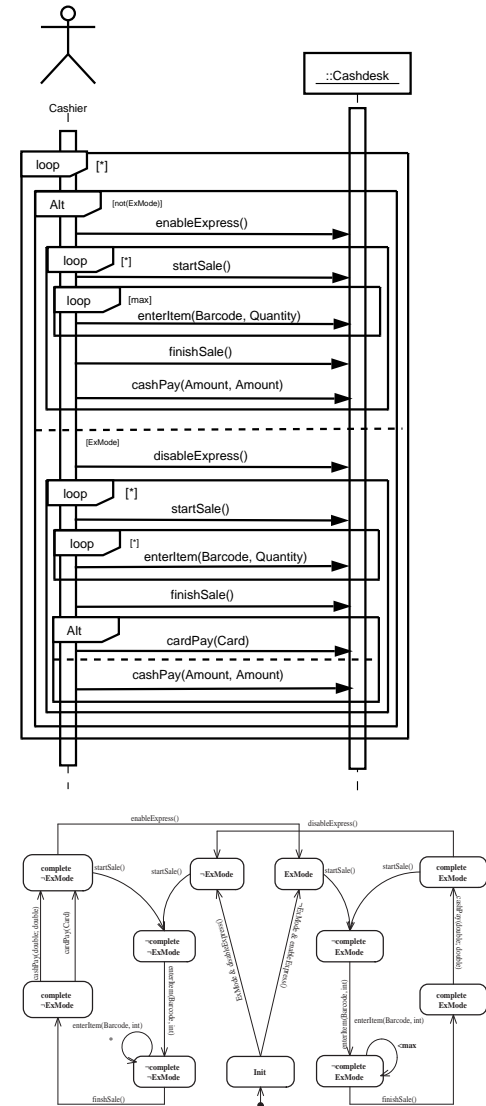
Applying rCOS methodology to CoCoME:

- input: informal problem description
- use case-driven model of requirements using diagrams and UTP
- refinement to code
- from objects to components
- formal verification/analysis
- runtime assurance of properties

Formalised Requirements

- Sequence diagrams (interaction with Actor)
- State diagrams (control flow in Use Case Controller class)
- Regular trace for rCOS component contracts
- Functional specifications of operations (pre/post)

⇒ Separation of control and data



Functional Specifications

Object-oriented rCOS:

- Pre-/Post conditions
- Invariants
- Class definition

```
class      C [extends D] {
attributes  $T x = d, \dots, T_k x = d$ 
methods    $m(T \text{ in}; V \text{ return}) \{$ 
           pre:           $c \vee \dots \vee c$ 
           post:          $\wedge (R; \dots; R) \vee \dots \vee (R; \dots; R)$ 
                        $\wedge \dots \dots$ 
                        $\wedge (R; \dots; R) \vee \dots \vee (R; \dots; R) \}$ 
           .....
```

```
            $m(T \text{ in}; V \text{ return}) \{ \dots \dots \}$ 
invariant  $Inv$ 
           }
```

CoCoME Example

Use Case **UC 1: Process Sale**

class *Cashdesk*

method *enterItem(Barcode c, int q)*

```
pre: /* there exists a product with the input barcode c */
store.catalog.find(c) ≠ null
post: /* a new line is created with its barcode c and quantity q, and then */
line' = LineItem.New(c/barcode,q/quantity)
/* the subtotal of the line item is set, and then */
; line.subtotal' = store.catalog.find(c).price × q
/* add line to the current sale */
; sale.lines.add(line)
```

invariant *store* ≠ null ∧ *store.catalog* ≠ null ∧ *sale* ≠ null

Checking Consistency of Specifications

- Static consistency (think “*compiler*”):
 - all types and methods are defined
 - type checking of signatures
 - consistency with class diagram
- Dynamic consistency:
 - sequence diagram implements protocol required by state diagram
 - regular trace is abstraction of diagrams
 - application dependent properties

Dynamic checking: e.g. through FDR

Navigation path in method $C.m()$:

$$C :: m() \{ \begin{array}{l} c(a_{11} \dots a_{1k_1} \cdot x_1, \dots, a_{\ell 1} \dots a_{\ell k_\ell} \cdot x_\ell) \\ \wedge \quad le' = e(b_{11} \dots b_{1s_1} \cdot y_1, \dots, b_{t1} \dots b_{ts_t} \cdot y_t) \end{array} \}$$

Refinement:

$$\begin{aligned} C :: & \quad check() \{ return' = c(a_{11} \cdot get_{\pi_{a_{11}}} x_1(), \dots, a_{\ell 1} \cdot get_{\pi_{a_{\ell 1}}} x_\ell()) \} \\ & \quad m() \{ \text{if } check() \text{ then } r_1 \cdot do\text{-}m_{\pi_{r_1}}(b_{11} \cdot get_{\pi_{b_{11}}} y_1(), \\ & \quad \quad \quad \dots, b_{s1} \cdot get_{\pi_{b_{s1}}} y_s()) \} \\ T(a_{ij}) :: & \quad get_{\pi_{a_{ij}}} x_i() \{ return' = a_{ij+1} \cdot get_{\pi_{a_{ij+1}}} x_i() \} \quad (i : 1..l, j : 1..k_i - 1) \\ T(a_{ik_i}) :: & \quad get_{\pi_{a_{ik_i}}} x_i() \{ return' = x_i \} \quad (i : 1..l) \\ T(r_i) :: & \quad do\text{-}m_{\pi_{r_i}}(d_{11}, \dots, d_{s1}) \{ r_{i+1} \cdot do\text{-}m_{\pi_{r_{i+1}}}(d_{11}, \dots, d_{s1}) \} \\ & \quad \text{for } i : 1..f - 1 \\ T(r_f) :: & \quad do\text{-}m_{\pi_{r_f}}(d_{11}, \dots, d_{s1}) \{ x' = e(d_{11}, \dots, d_{s1}) \} \\ T(b_{ij}) :: & \quad get_{\pi_{b_{ij}}} y_i() \{ return' = b_{ij+1} \cdot get_{\pi_{b_{ij+1}}} y_i() \} \quad (i : 1..t, j : 1..s_i - 1) \\ T(b_{is_i}) :: & \quad get_{\pi_{b_{is_i}}} y_i() \{ return' = y_i \} \quad (i : 1..t) \end{aligned}$$

Object-oriented Design

From functional specifications to code:

- rCOS allows big-step refinement
- provably correct rules/design patterns
- often directly realizable in high-level programming languages:

```
updateInventory()
```

```
Class Cashdesk:
```

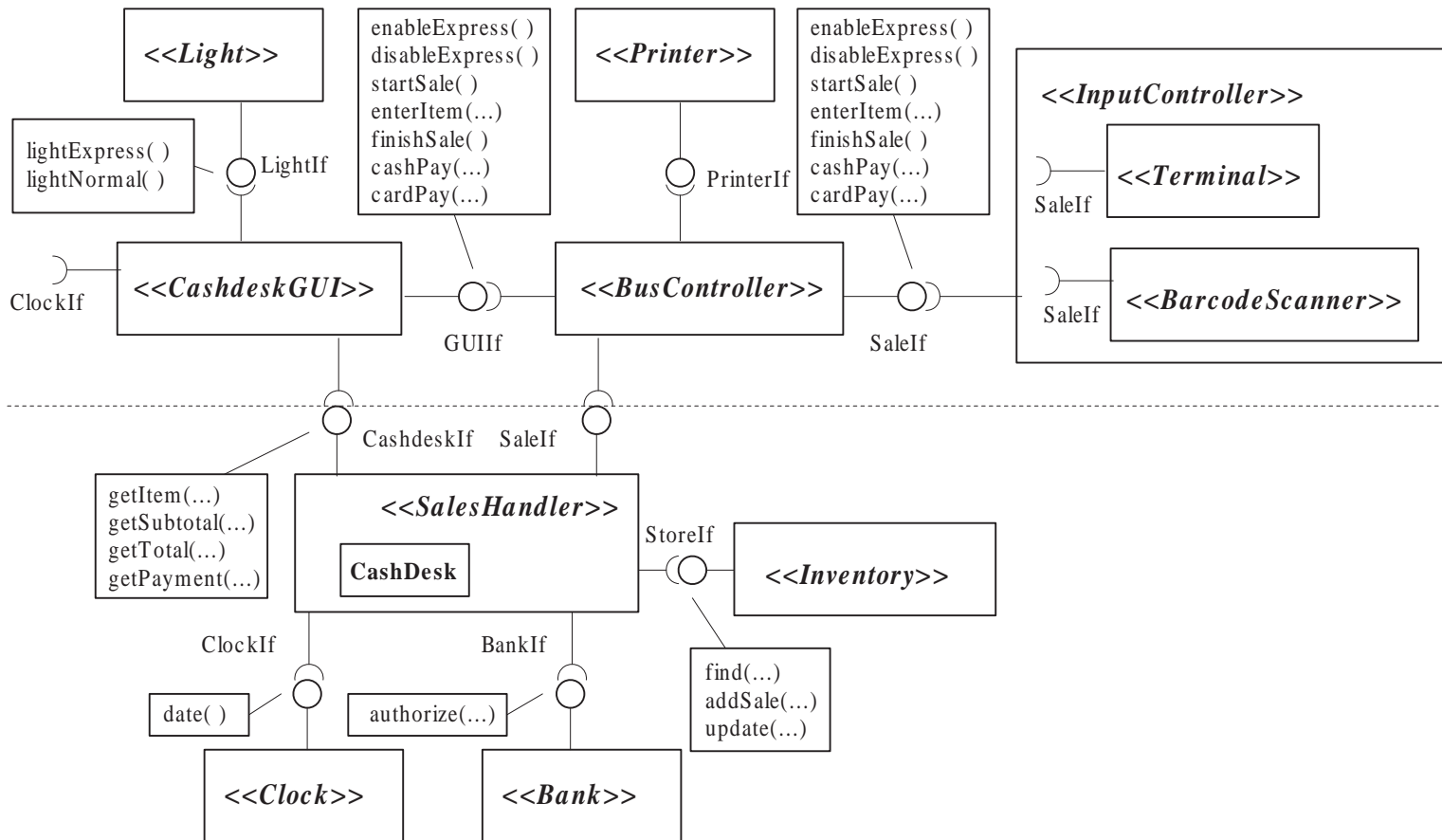
```
 $\forall l \in \text{sale.lines}, p \in \text{store.catalog} \bullet ( \quad \text{if } p.\text{barcode}=l.\text{barcode} \text{ then}$   
 $\quad \quad p.\text{amount}' = p.\text{amount} - l.\text{quantity} )$ 
```

yields almost executable Java with assertions:

```
class Product:      update(int qty) { amount := amount-qty }  
class set(Product): update(Barcode code, int qty) {  
    Iterator i := iterator();  
    while (i.hasNext()) {  
        Product p := i.next();  
        if p.barcode=code then p.update(qty); }  
class Store:      update(Barcode code, int qty) { catalog.update(code,qty) }
```



Component Diagram



Components, rCOS View

```
define Salelf { enableExpress(), disableExpress(), startSale (), enterItem (..),  
                finishSale (), cardPay(..), cashPay(..) }
```

component Terminal

required interface Salelf

```
protocol { ([disableExpress!] startSale! enterItem!* finishSale! [cardPay! | cashPay!])* }
```

component SalesHandler

required interface Clocklf { date() }

required interface Banklf { authorize (..) }

required interface Storelf { update (..), find (..), addSale(..) }

provided interface Salelf

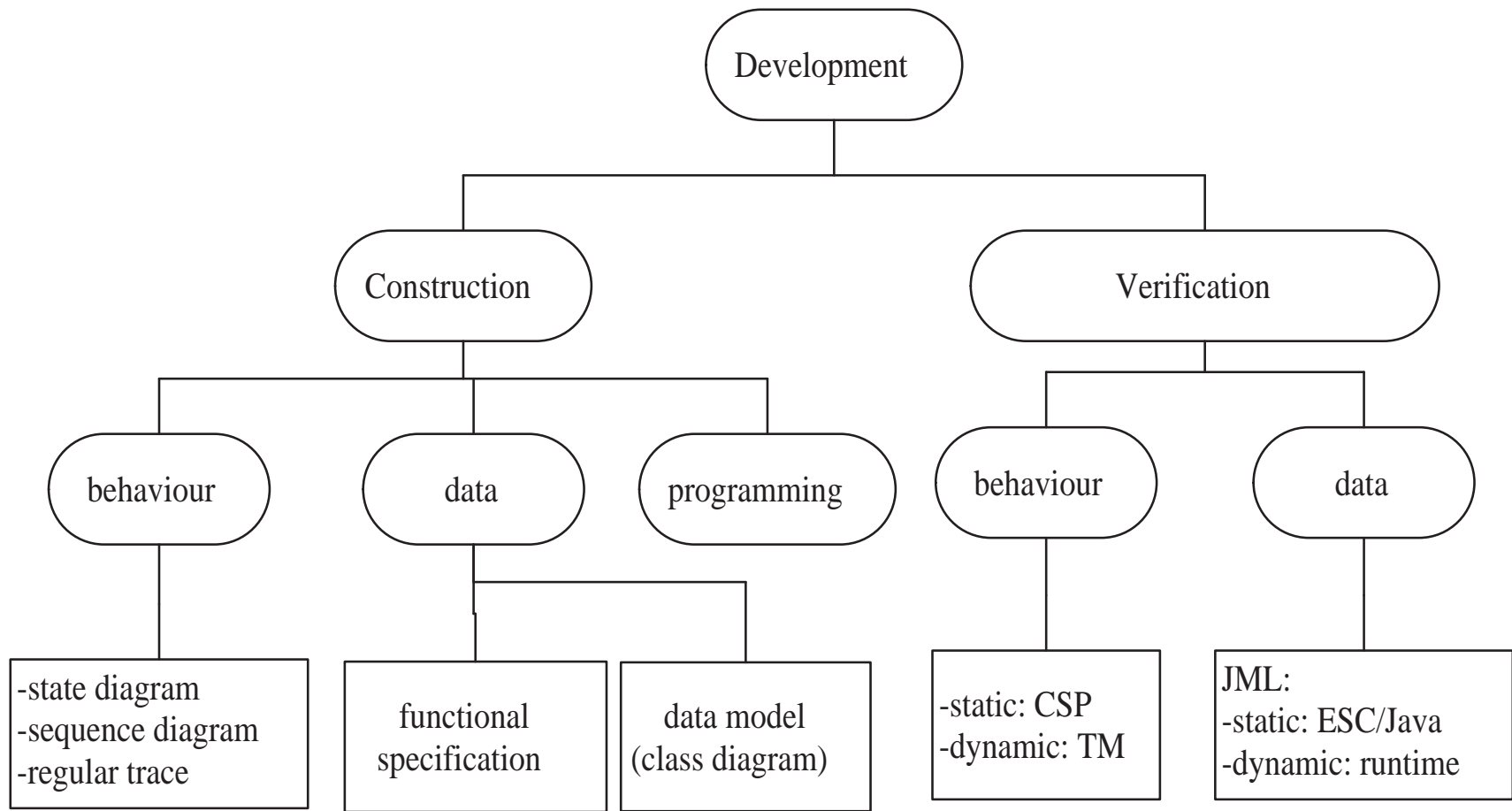
provided interface Cashdesklf { getItem(..), getSubTotal (..), getTotal (..), getPayment() }

```
protocol { ( [ ?enableExpress ( ?startSale date! (?enterItem find !)(max) ?finishSale  
                ?cardPay authorize! addSale!)*  
            | ?disableExpress ( ?startSale date! (?enterItem find !)* ?finishSale  
                [ ?cardPay authorize! addSale! update!*  
                | ?cashPay addSale! update!* ] )* )*
```

class *Cashdesk* **implements** Salelf, Cashdesklf



CoCoME Toolchain



Roles in the Development Process

Different tasks require different expertise:

- data/application domain expert
(object/class diagrams)
- specification experts:
 - dynamic behaviour (sequence/state diagrams)
 - functional specification (mathematical modelling)
- verification engineer:
 - behaviour: CSP
 - data: JML
- programmer

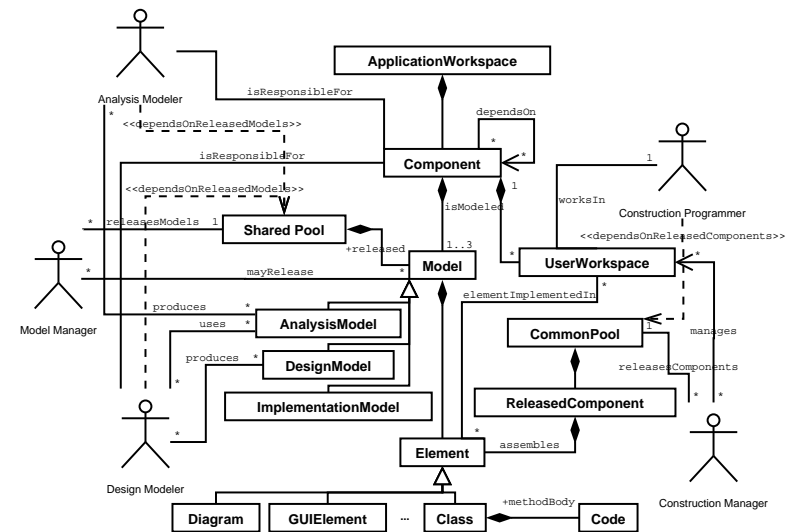
Comparison with MasterCraft

What is MasterCraft?

- developed by Tata Research Development & Design Centre
- covers entire software development life-cycle: requirements gathering and analysis, implementation, testing, deployment, maintainance

roles:

- Analysis Modeler
- Design Modeler
- Model Manager
- Construction Programmer
- Construction Manager



The Team

- Team Leader: Zhiming Liu (rCOS)
- Dang Van Hung, *Naijun Zhan*, *Qu Nan* (QoS analysis)
- *Anders P. Ravn* (rCOS)
- Volker Stolz (consistency, CSP)
- Fellows:
 - Hakim Hannousse, Joseph Okika, Lu Yang (specification)
 - Liu Yang (specification, implementation)
 - Chen Zhenbang (JML modelling)

Effort

- 1 day for initial outline of use case specification
- 4-man week for remaining use cases
- 1 day for sample refinement of first use case
- 4-man for remaining use cases
- one week of experimenting with CSP/FDR to get first results
- generated 65 classes / 4000 LoC in Java

Only for business logic, does not yet include:

- GUI
- middle ware
- glue code

Requirements Elicitation Takes Time!

Where did most of the time go?

- a lot of time spent on synchronizing with ongoing changes as the requirements stabilize
- even more time necessary to ensure that specifications are actually consistent (implementing and checking the diagrams in a specification language like CSP)
- MOST of the time: discussing HOW to make a change

Advantage of *separation of concerns*:

- most problems related to control flow, not data
⇒ different aspects of single use case tackled in parallel

CoCoME specification only semi-formal rCOS:

- no machine-readable notation
- no automation/tool-support yet
- formal “pen-and-paper” proofs impractical for even medium-sized systems that keep evolving

On the practical side:

- how to model *middle ware* and *deployment*?
- currently unmanaged *glue* code

Closing the Semantic Gap

Desirables:

- managed specifications within single framework
- assure static consistency
- (correct) translation into input for respective tools
 - refinement (patterns, QVT)
 - software (business logic AND runtime checking)
 - verification (CSP, JML)
- interpret/visualize verification results

⇒ framework for rCOS methodology

Required tools:

- machine-readable syntax, foundation for tools (Eclipse Modeling Framework EMOF? DSLs?)
- static consistency/wellformedness checker
- PVS prover (Aalborg, DK)
- dynamic consistency checker (CSP/FDR)
- model-driven development-tool support through transformations (QVT)
- generate runtime assurance: assertions, trace properties
- back end: Java, Spec#

Application of rCOS in case study:

- methodology separates data from control
- driven by use cases
- ensures consistency of specification
- provably correct refinement steps into OO code
- assertions based on formal specification

To do:

- automate development / tool support
- formally model middle ware

Bibliography

- **Harnessing Theories for Tool Support,**
Z. Liu, V. Mencl, A. P. Ravn, L. Yang; ISoLA 2006
- **Automating Correctness Preserving Model-to-Model Transformation in MDA,**
L. Yang, V. Mencl, V. Stolz, Z. Liu; AWCVS 2006
- **Separation of Concerns and Consistent Integration in Requirements Modelling,**
X. Chen, Z. Liu, V. Mencl; SOFSEM 2007.
- **A Refinement Driven Component-based Design,**
Z. Chen, Z. Liu, V. Stolz, L. Yang; ICECCS 2007.
- **CoCoME** book chapter, to be published as LNCS