

rCOS Method of Component-Based Model Driven Design

Zhiming Liu

<http://rcos.iist.unu.edu>



United Nations
University

UNU-IIST

International Institute for
Software Technology



Background

- Software complexity and correctness are handled by
 - factoring a model of a component into models of different views of the **data computation**, **interaction protocol**, the **reactive behaviour**,
 - techniques for model **analysis**, **verification** and **correct by construction** through model transformations,
 - tool support for **model construction**, **model verification** and **model transformations**.
- **Four strands**: theory, tool support, experiments and technology transfer.
- **Ambition**: achieve this aim by **developing** and **teaching** a coherent and comprehensive methodology that begins with design for verification and validation and integrates verification into development.



Key Features and Concepts in rCOS

- use-case driven development, use of oo design patterns
- **Interface contracts**: constraints on the environment of component
- **Components**: implements contracts, with contracts of **provided** and **required interfaces**
- **Refinement**: refinement of interface contracts and components
- **Compositions**: renaming, restriction, hiding, parallel, plugging, synchronisation
- **Classes and Objects**: link object and component systems

Has a semantic root of Hoare&He's UTP

Contract Supports Incremental Modelling

Specifies what is needed for a component to be *used* in building and maintaining a software without the need to know its design

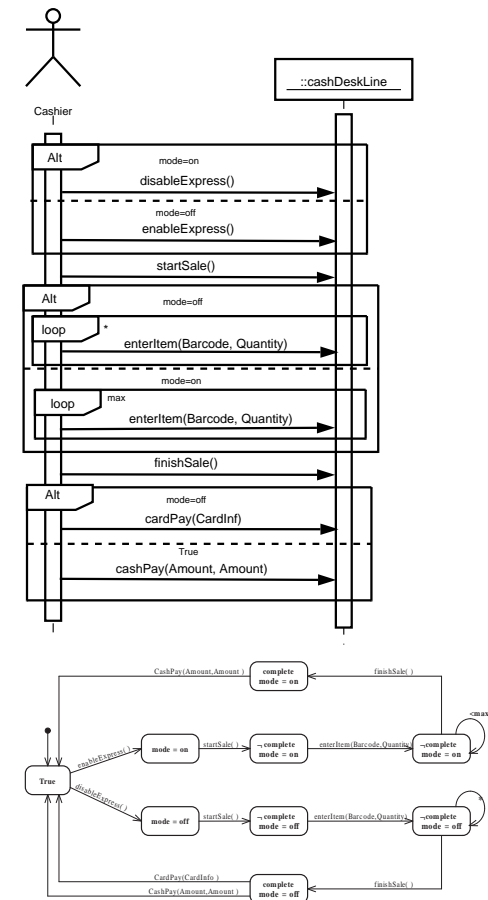


- **static functionality:** $p \vdash R$
- **interaction protocol:** $\text{traces} \subseteq \{put(x), get(y) : x : T\}^*$
- **timing** $[l, b]$ or $[s, m, l, b]$
- **location, address, resources, and general QoS**

Separation of concerns and consistency of different views

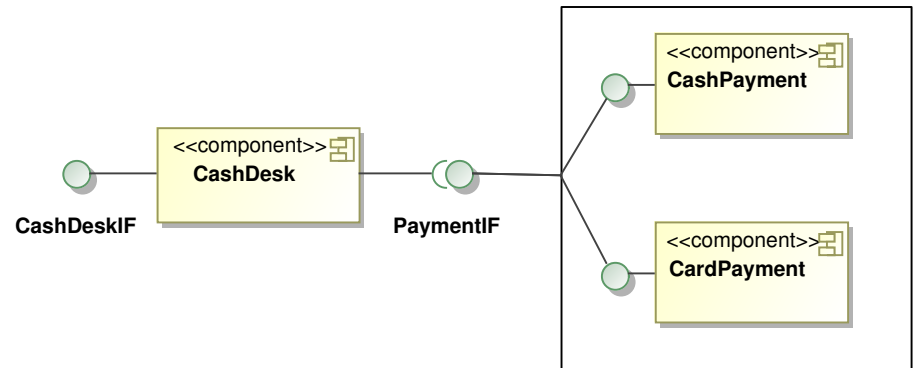
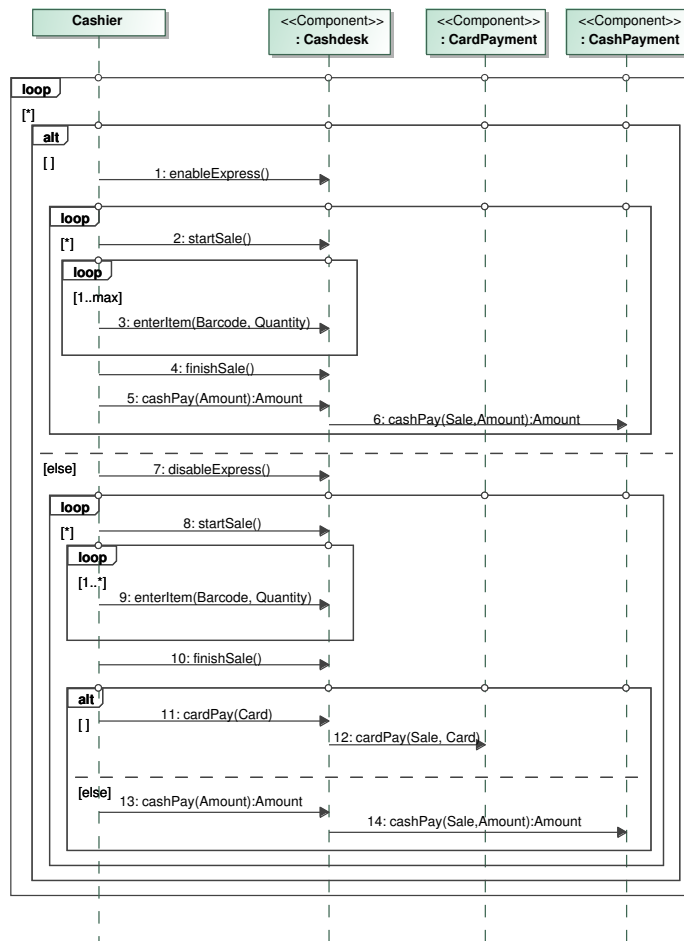
RA: Use-Case vs Contracts of Component

- sequence diagram (actor's behavior)
- state diagram (component's reactive behavior)
- class diagrams (data and object structure)
- CSP processes *Actor || Comp*
- functionality actions $m() \{ f : p \vdash R \}$
- $Ctr = (I, Spec, Prot)$
- UML models translated to and from rCOS



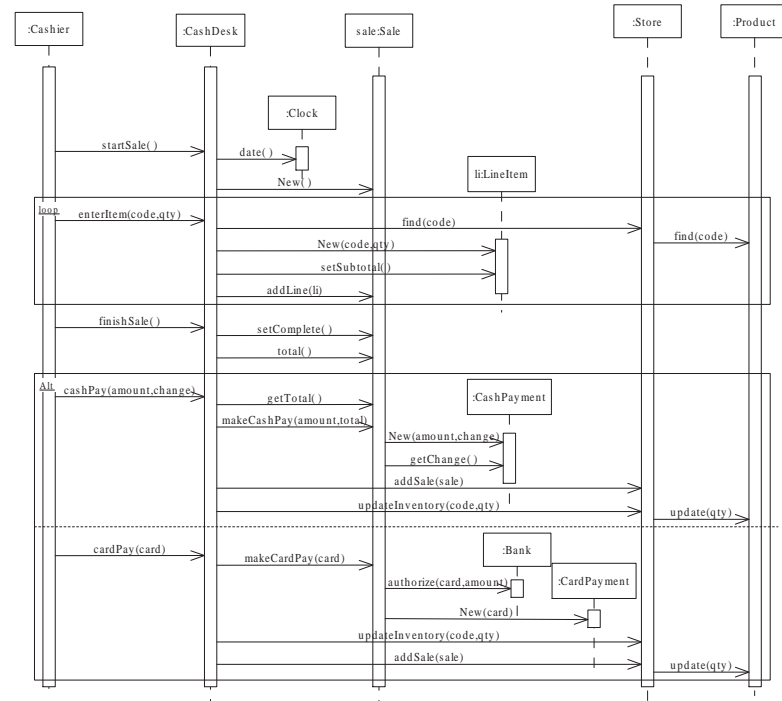
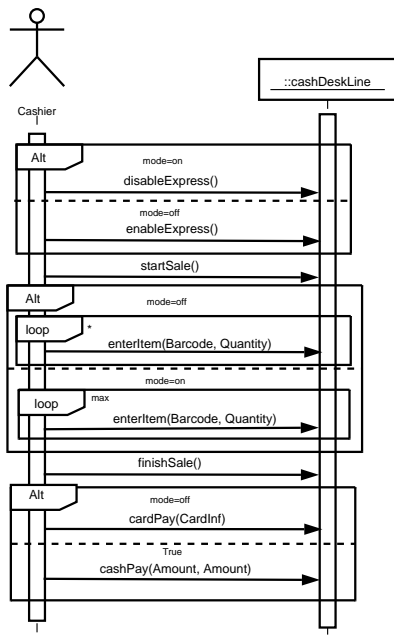
use cases = composition of specified componen

$ProcessSale \triangleq CashDesk \ll (CashPay || CardPay)$



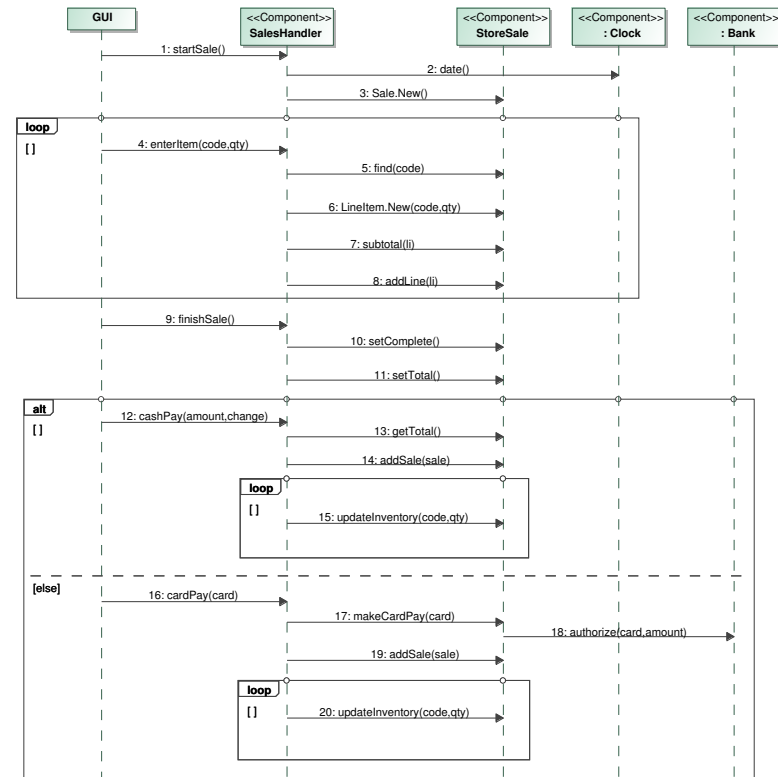
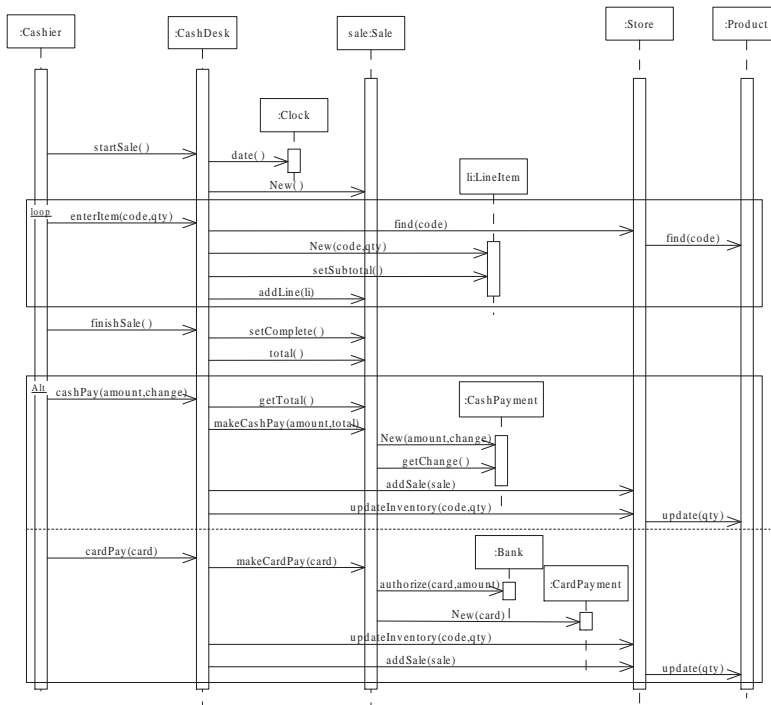
Design: OO Refinement & Design Patterns

Expert pattern for functional decomposition



- automated design patterns
- refactoring rules
- generating proof obligations

Design: CB Decomposition and Composition



● $ProcessSale \triangleq SalesHandler \ll (StoreSale \parallel Clock \parallel Light)$

● abstract oo interfaces to component interfaces

● replace oo interaction to platform middlewares

GUI and Hardware Controllers

- GUI objects, controllers of hardware (barcode scanners, card readers printers, cash boxes) and application components interact
- Established event-based modelling and verification, such as CSP and FDR

Conclusion: <http://rcos.iist.unu.edu>

