Service-Oriented Modeling of CoCoME with FOCUS and AutoFOCUS

Florian Hölzl

Chair IV: Software and Systems Engineering

Department of Informatics

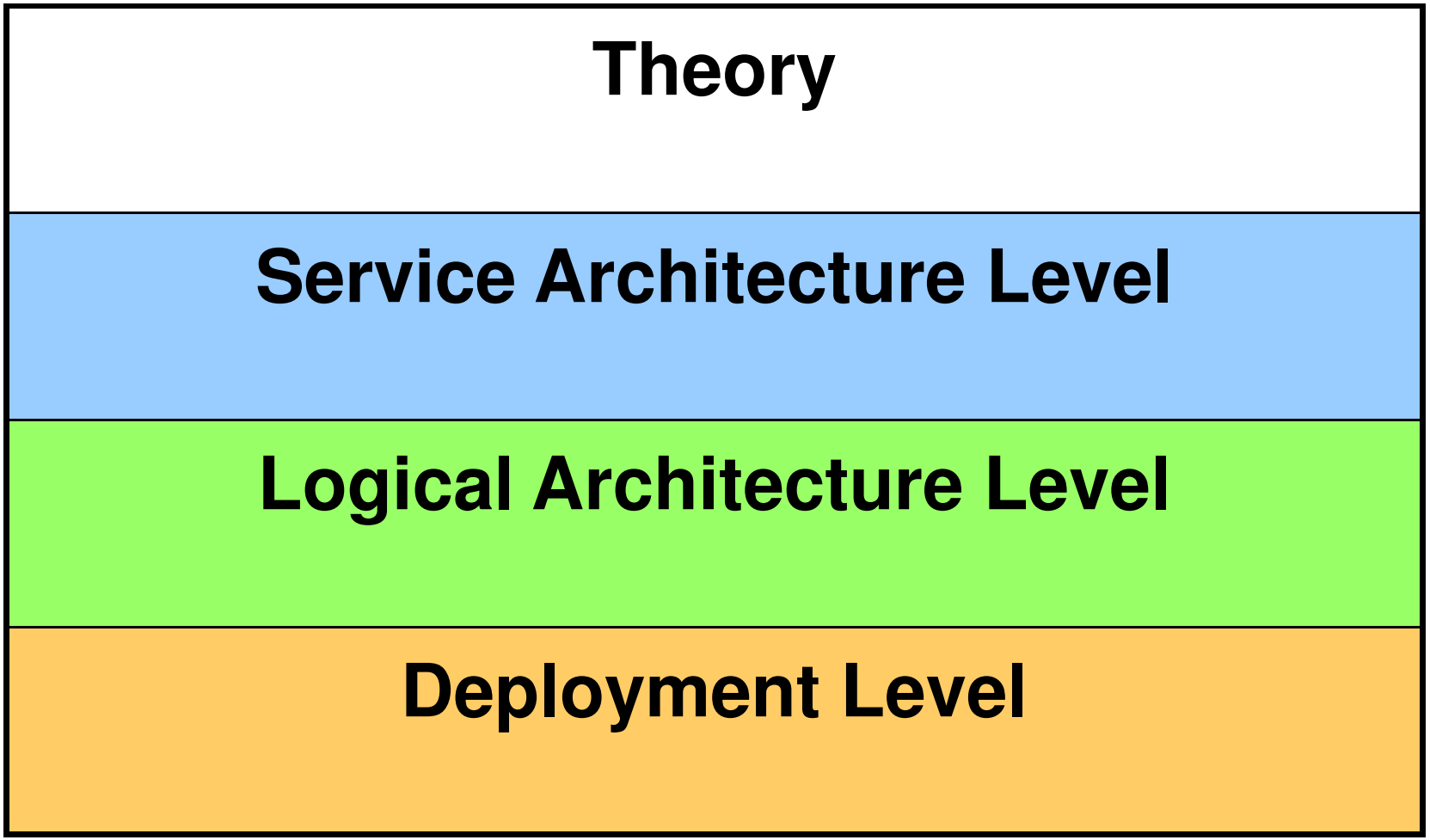
Technische Universität München

TECHNISCHE UNIVERSITÄT MÜNCHEN

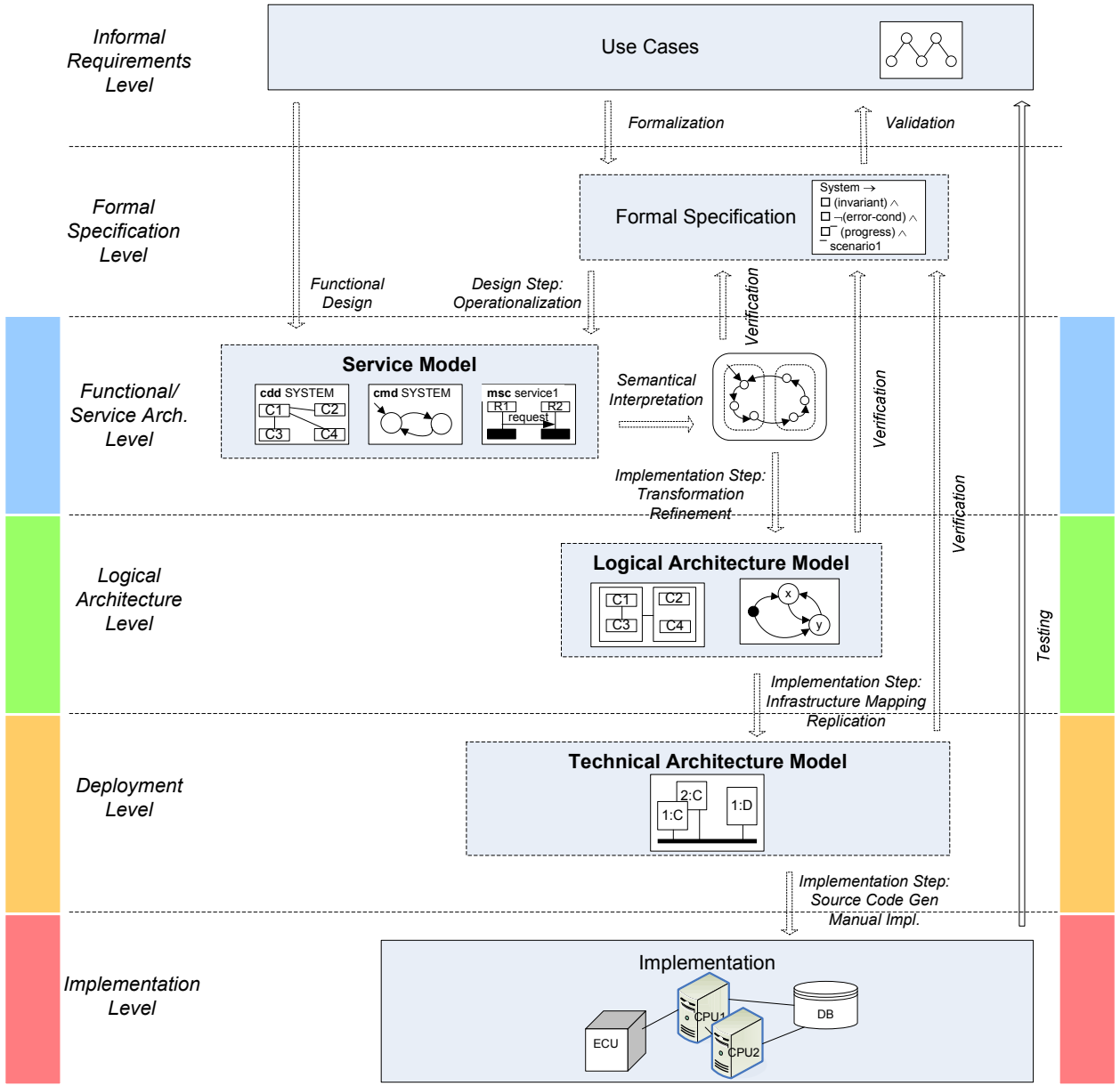
Our Team

Dr. Bernhard Schätz	Model-Based Software Development
Michael Meisinger	Service-Based Software Development
Sabine Rittmann	Service-Oriented Software Engineering
Doris Wild	Automotive Software and System Design
Maria Spichkova	Verification of Embedded Systems
Jorge Fox	Aspect Oriented Software Development
Dagmar Koss	Compatibility
Birgit Penzenstadler	Requirements Engineering for Subsystems
Marco Kuhrmann	Process Models
Florian Hölzl	Tool Support in Model-Based Development

Responsibilities / Contributions



Modeling Approach



TECHNISCHE UNIVERSITÄT MÜNCHEN

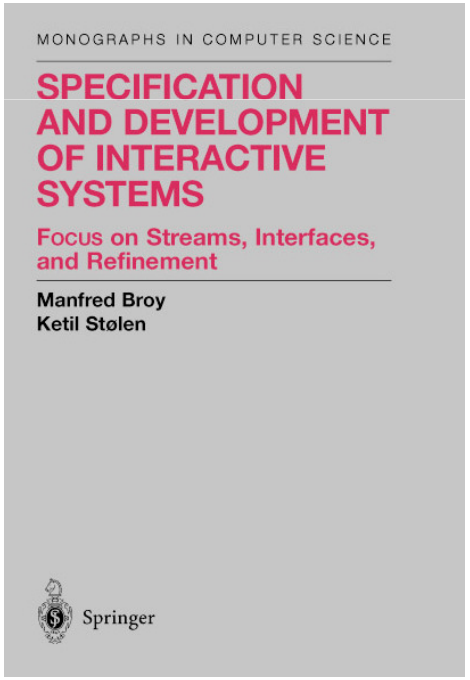
Part I: The FOCUS Component Model

System Model

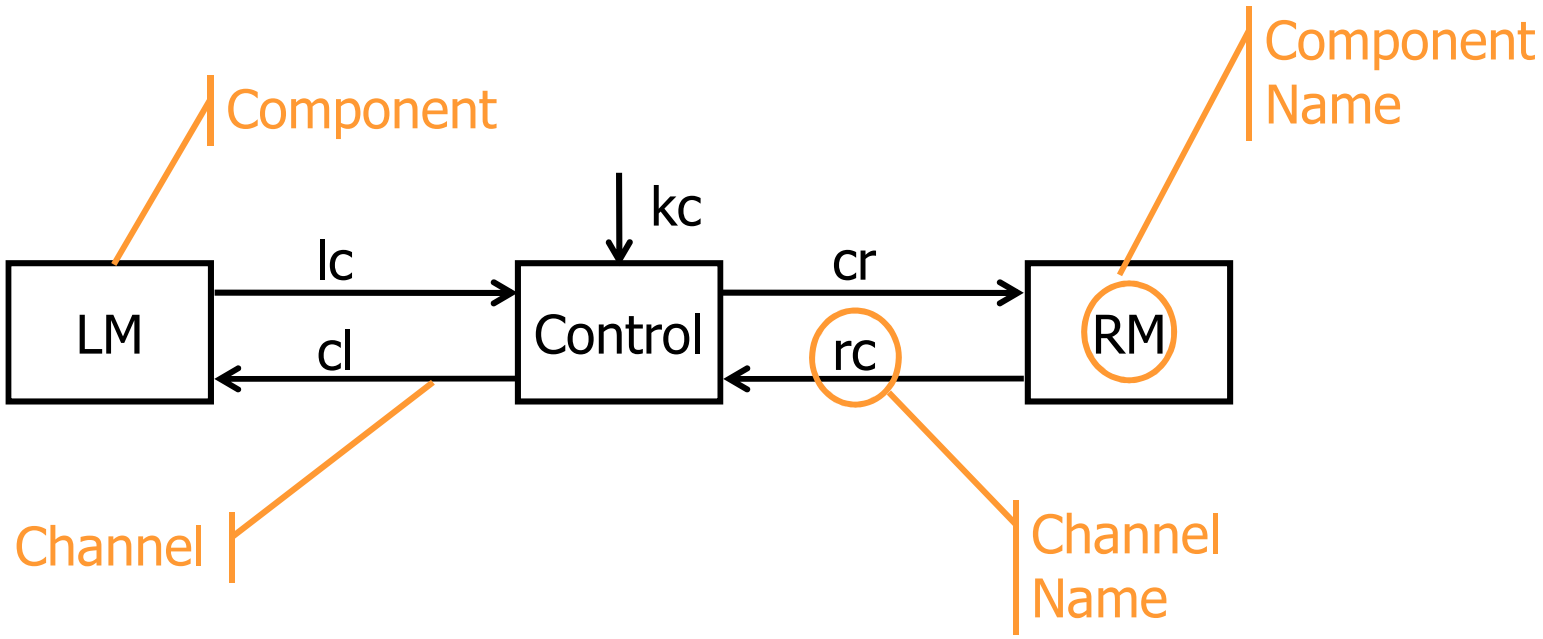
Streams

Strong Causality

Composition



System Model



System consists of

- named Components (with encapsulated States / State Machine)
- named Channel (possibly typed)

with a model of DISCRETE GLOBAL TIME

Modelling Channels: Streams

Terminology

- Channels connect two Subsystems or a System and its Environment (Input or Output Channels)
- Streams model Communication History of Channels
- Composed Systems are defined by Recursive Equations over Streams

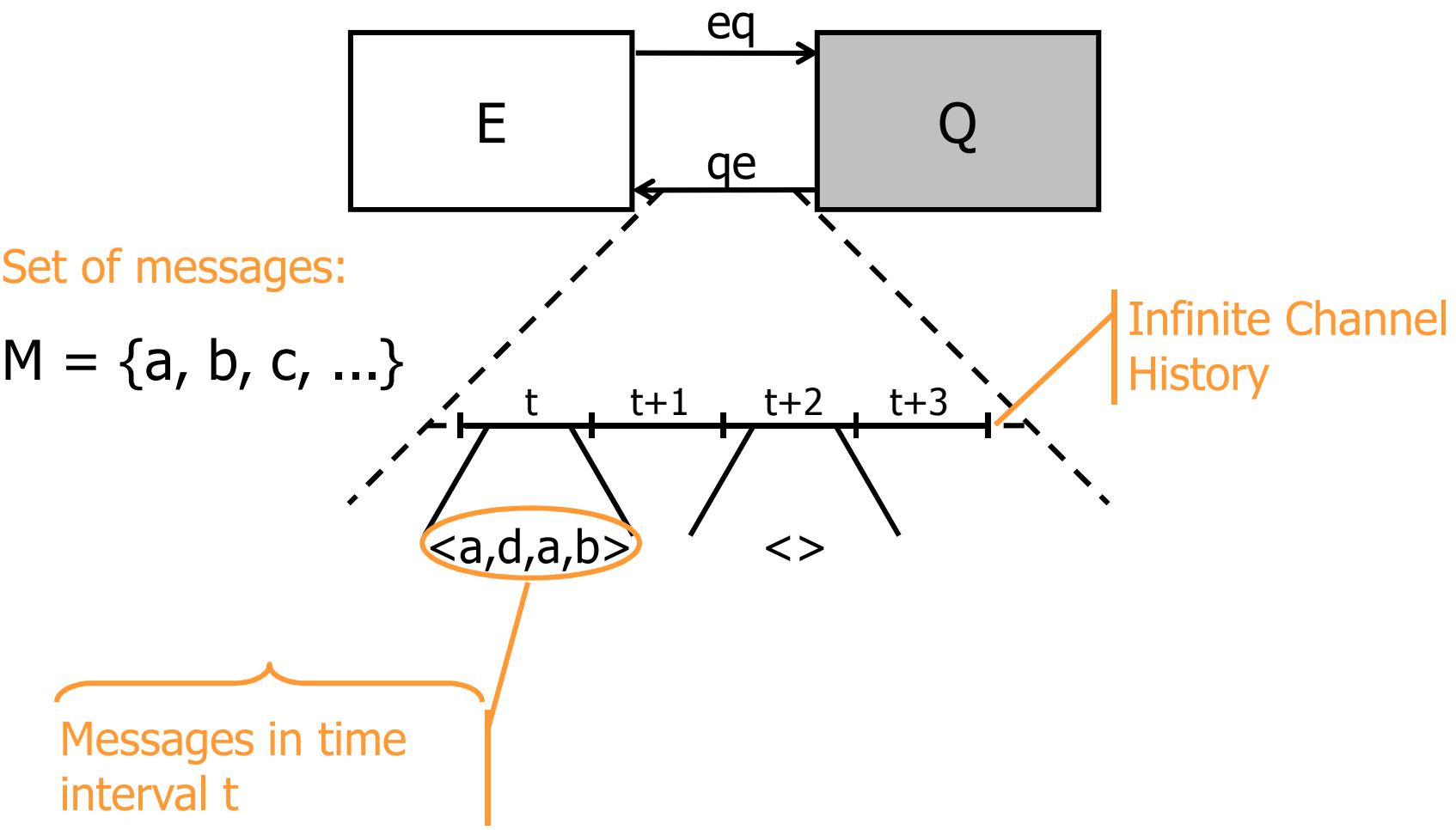
Streams

Let M be the set of messages

M^*	Set of finite sequences over M	
$\langle\rangle$	empty sequence	
M^∞	Set of infinite sequences over M	$\mathbb{N} \setminus \{0\} \rightarrow M$
M^ω	Set of streams: $M^\omega = M^* \cup M^\infty$	
$(M^*)^\infty$	Set of timed streams over M - a sequence of messages for each time interval	

Timed Streams

Semantic Model for Interface Behavior



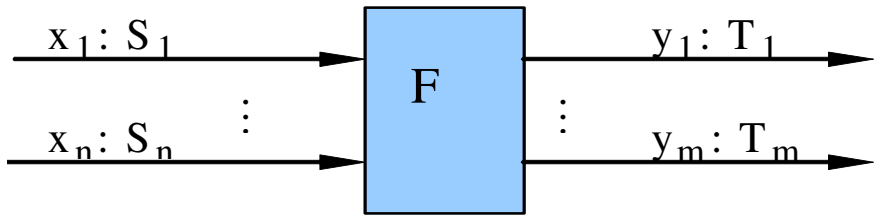
Interface Model

I = { x1, x2, …} set of input channels

O = {y1, y2, …} set of output channels

Interface Behavior: map input histories to output histories

$$F : \overrightarrow{I} \rightarrow \mathcal{P}(\overrightarrow{O})$$



Strong Causality

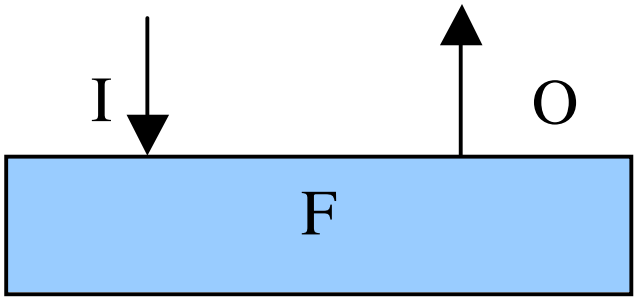
Interface Behavior $F : \overrightarrow{I} \rightarrow \mathcal{P}(\overrightarrow{O})$

Strong Causality

$$x, z \in \overrightarrow{I}, y \in \overrightarrow{O}, t \in \mathbb{N}$$

$$x{\downarrow_t} = z{\downarrow_t} \Rightarrow \{y{\downarrow_{t+1}} : y \in F(x)\} = \{y{\downarrow_{t+1}} : y \in F(z)\}$$

A causal component F is total,
e.g. F.x ≠ ∅ for all x,
OR F.x = ∅ für alle x

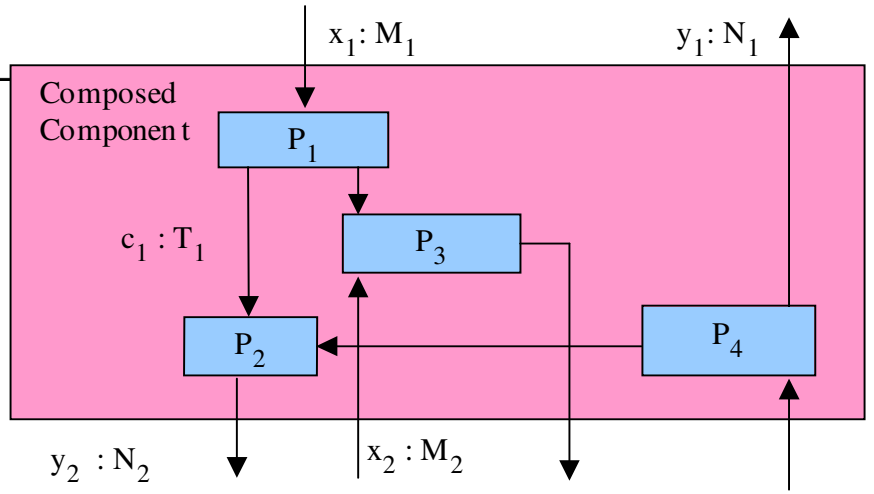


Composition of Specifications

in $x_1: M_1 , x_2: M_2 , ...$

out $y_1: N_1 , y_2: N_2 , ...$

$\exists\ c_1, c_2 , ... : P_1 \wedge ... \wedge P_n$



Part II: The CoCoME Model

Functional / Service Architecture

Logical Components Architecture

Deployment

Implementation

Functional / Service Architecture Level

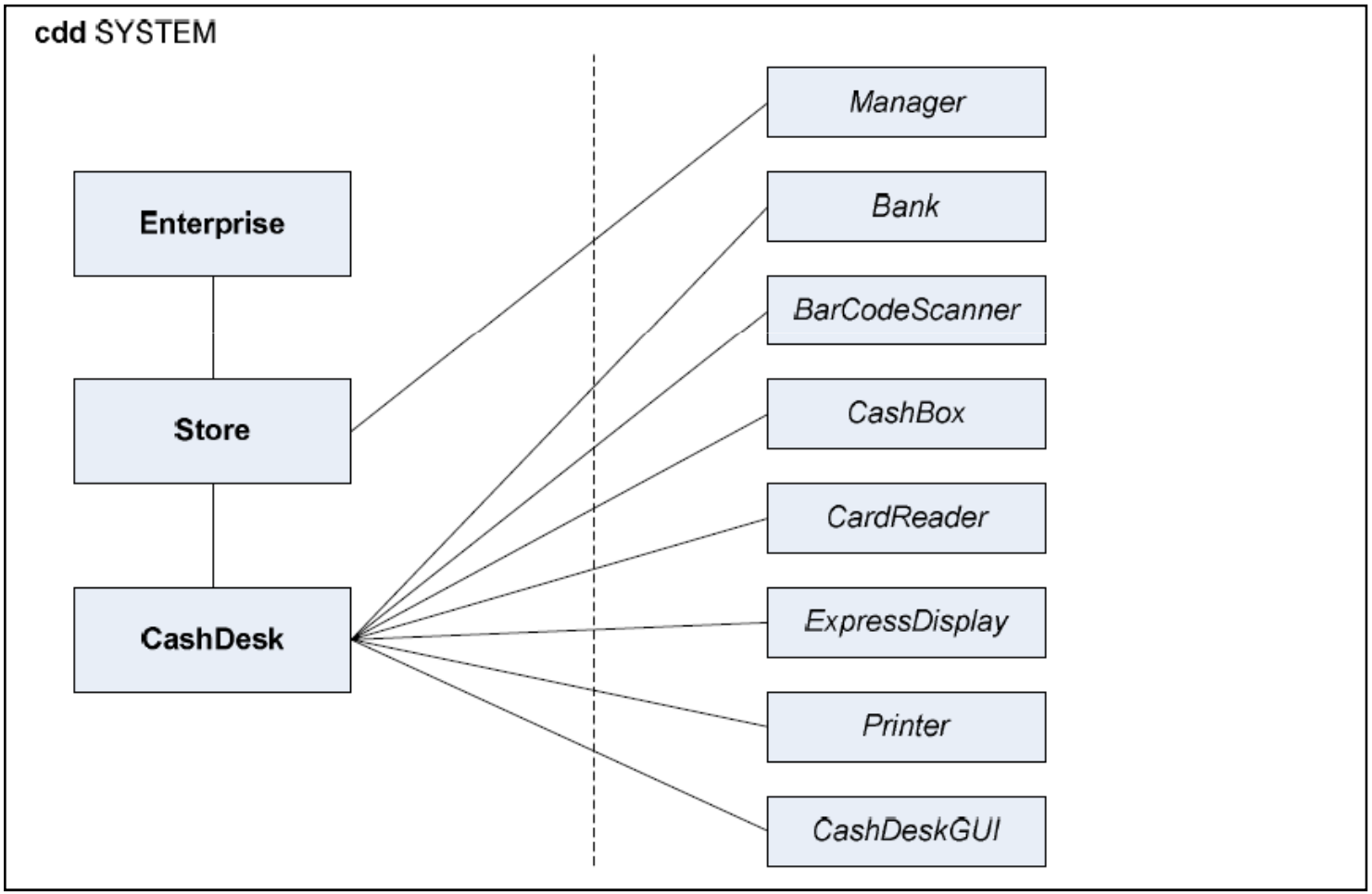
- Identify abstract components
- Identify communication dependencies
- Identify modes of operation of components
- Specify the services of each component as MSC
- Compose services using higher-level MSCs

- Refactor components, modes and services as needed

- Semantical Interpretation into Behavior Automata

Functional System Decomposition

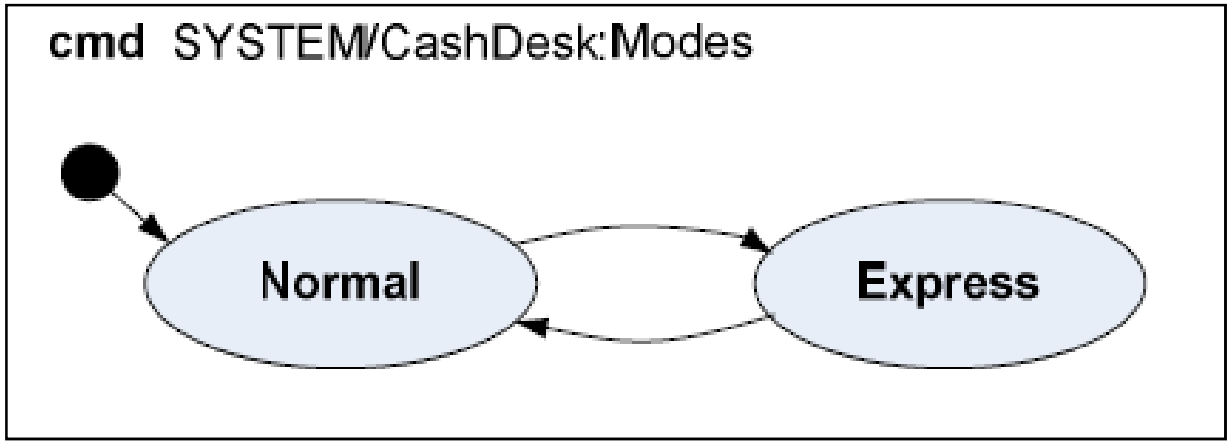
- System decomposes into communicating entities



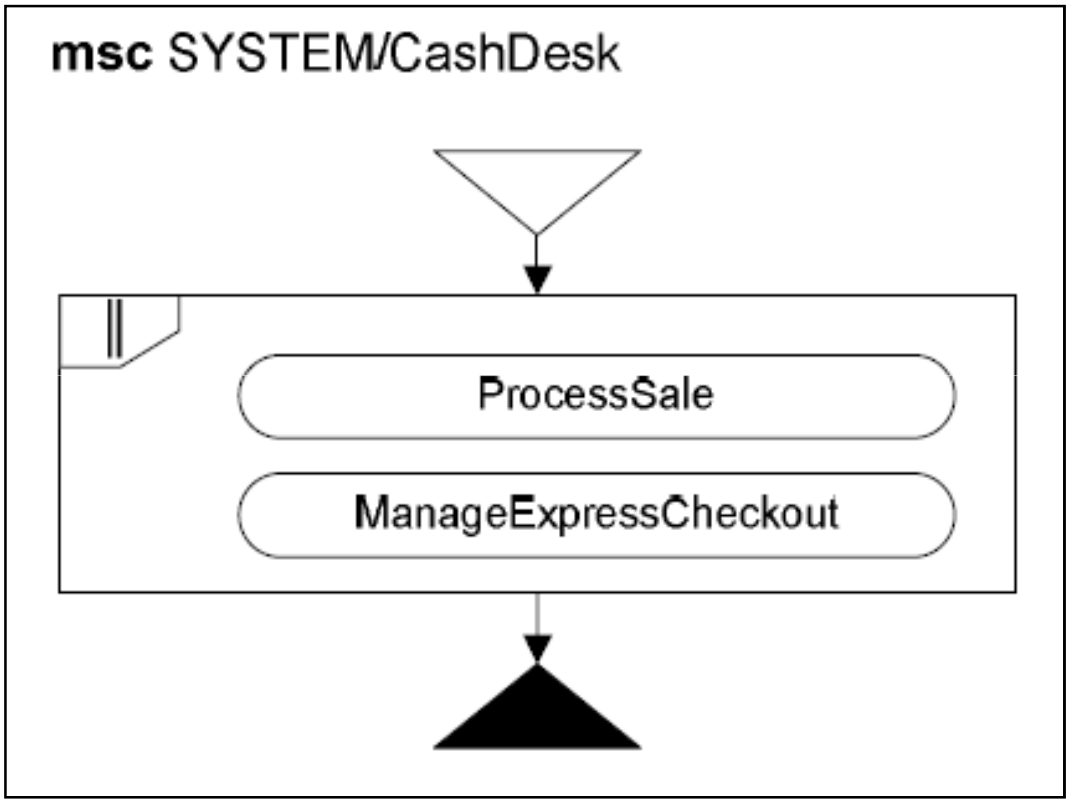
Department of Informatics
Chair IV: Software & Systems Engineering

TECHNISCHE UNIVERSITÄT MÜNCHEN

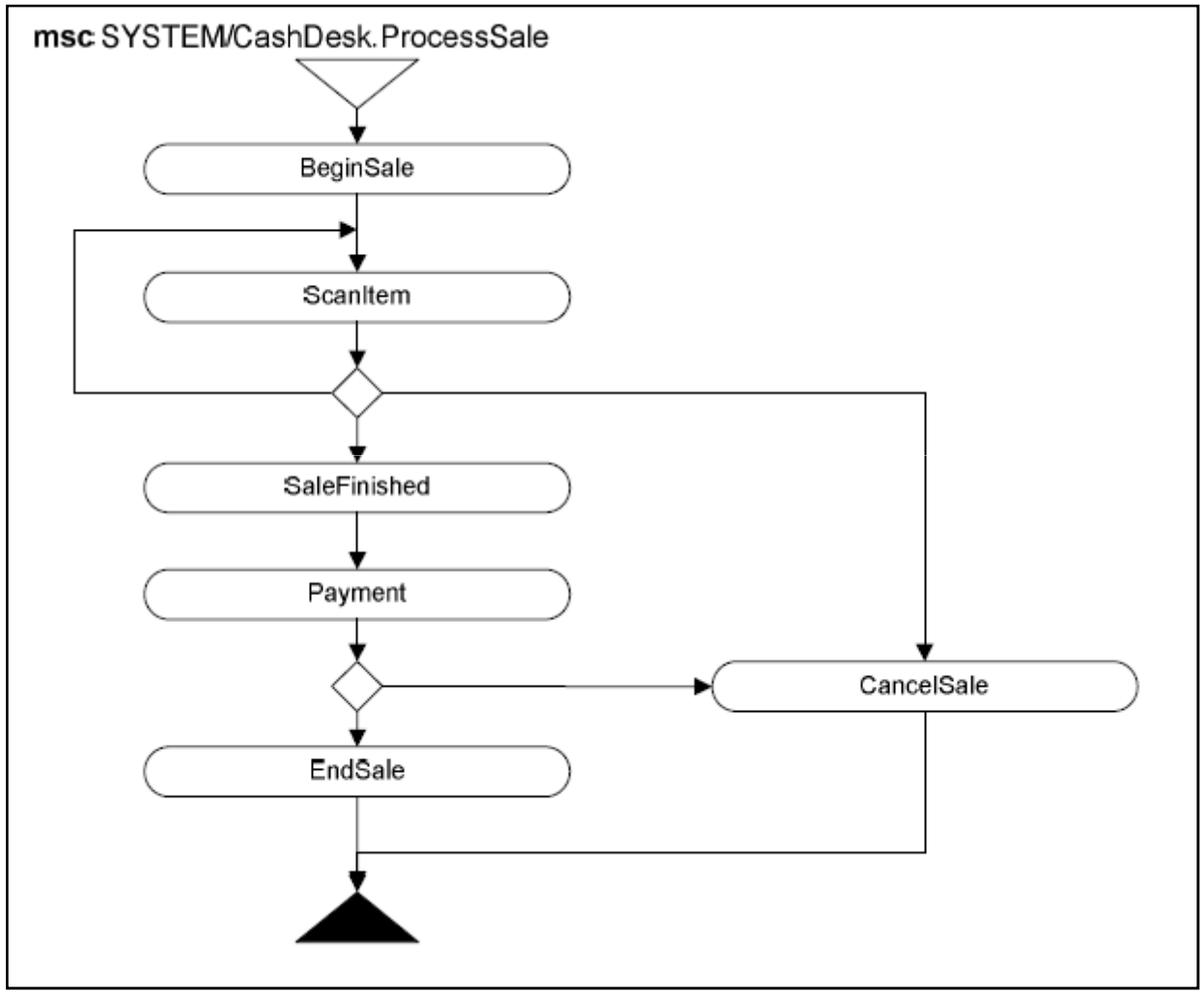
Component Mode Diagram



High-Level Message Sequence Chart

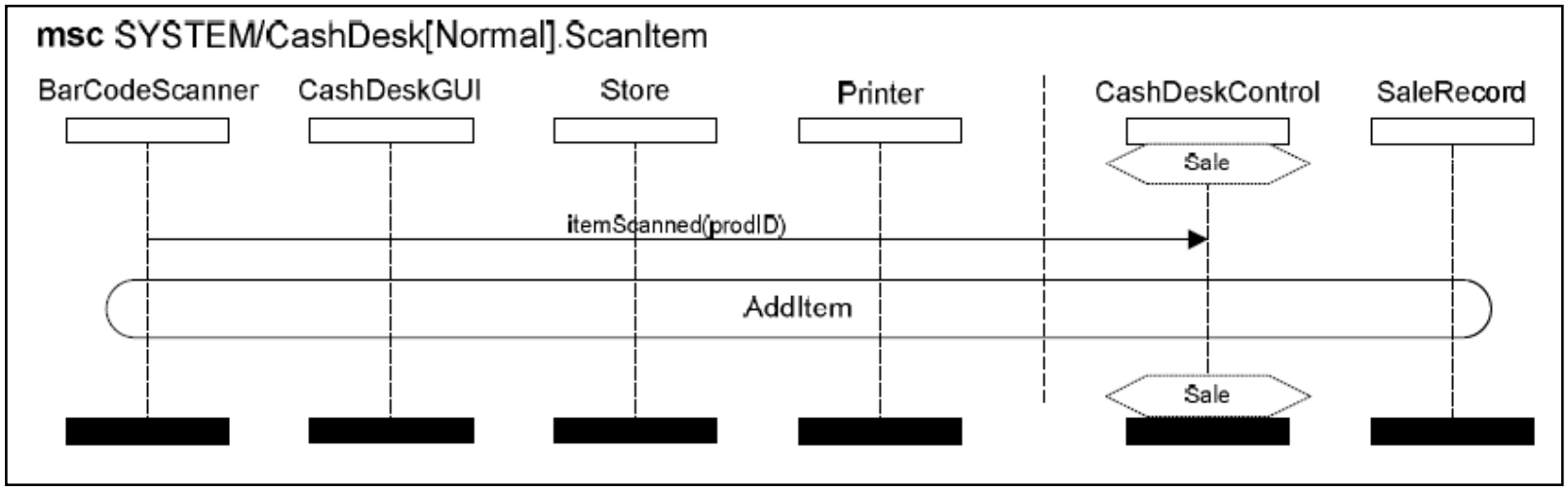


Hierarchical HMSC Decomposition

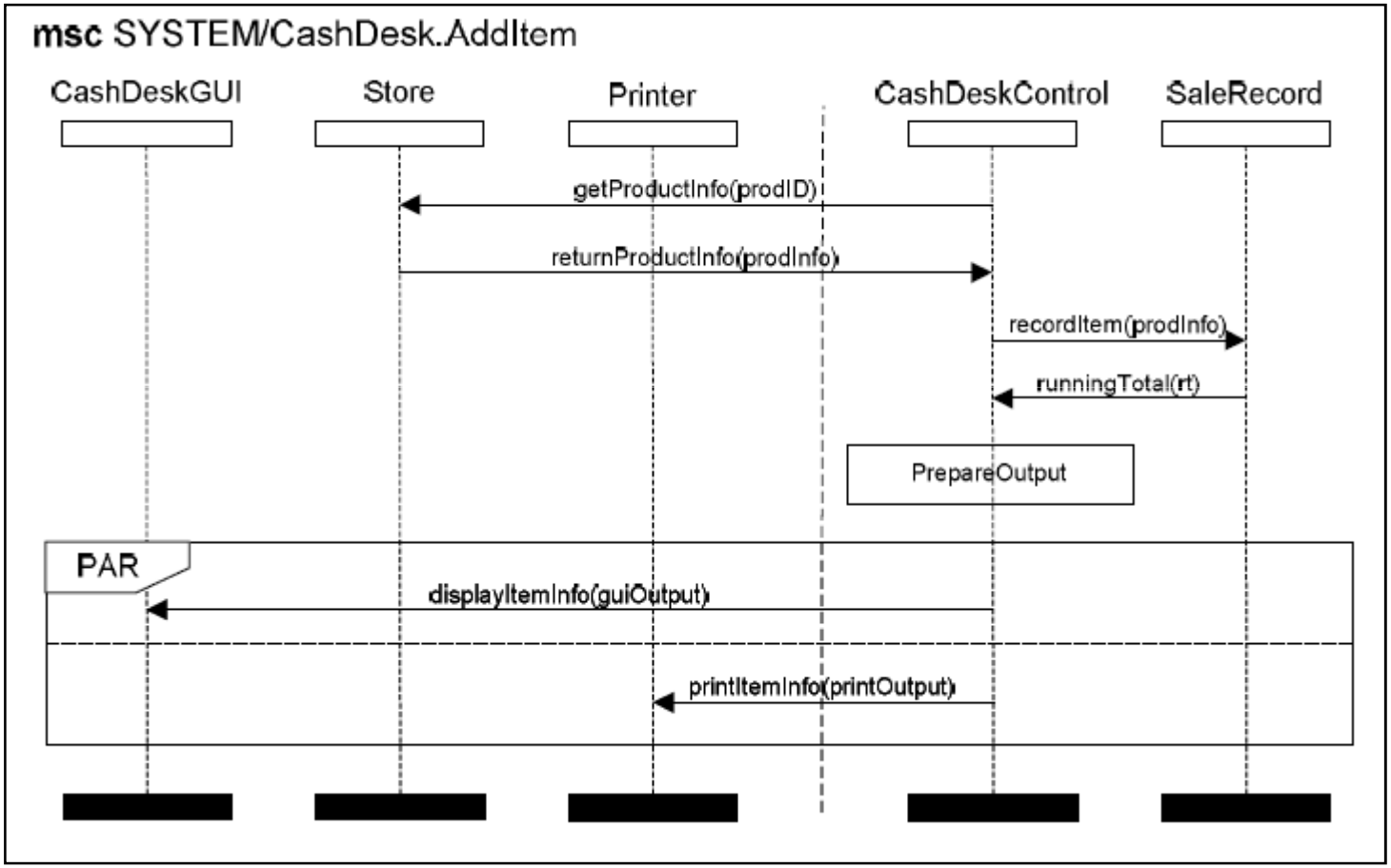


Department of Informatics
Chair IV: Software & Systems Engineering

Service Message Sequence Chart

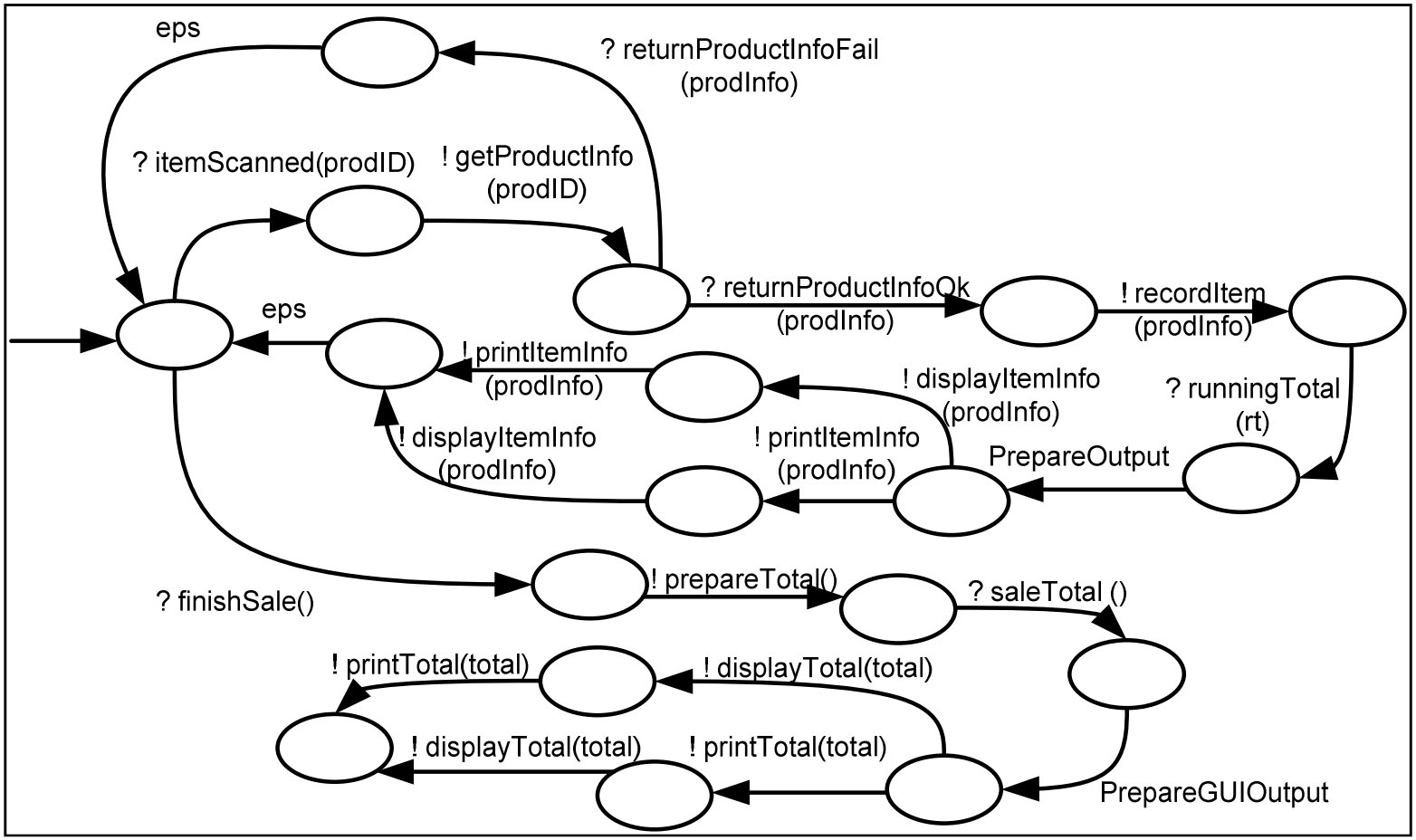


Subservice Message Sequence Chart



Semantical Interpretation

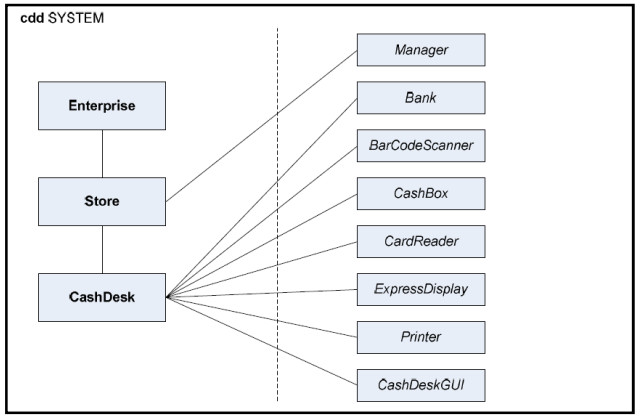
- Behavior Automata

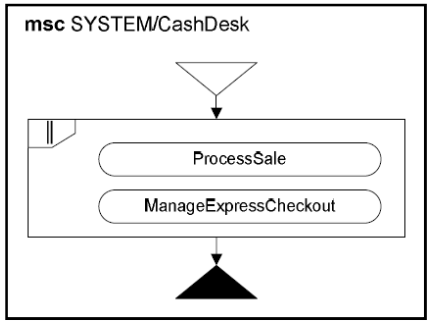


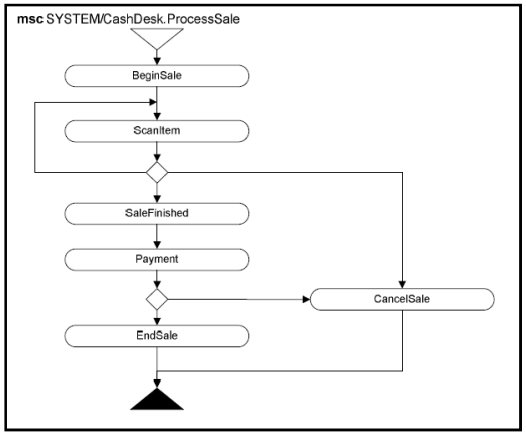
Functional Architecture Properties

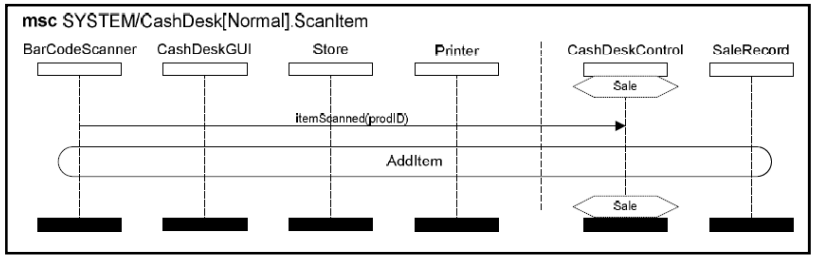
- Consistency
  - Service specification comply with mode switches
  - Interaction with externals comply with interface specification
- Completeness
  - Internal and external services have a service specifications
  - Service specification exist for each mode and mode transition
- Closed World Assumption
  - Complete consistent set of services form the exact specification of the components‘ behavior

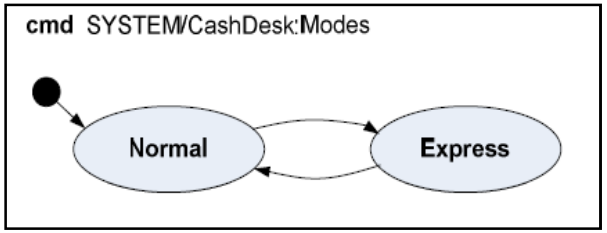
Functional Architecture Level Results

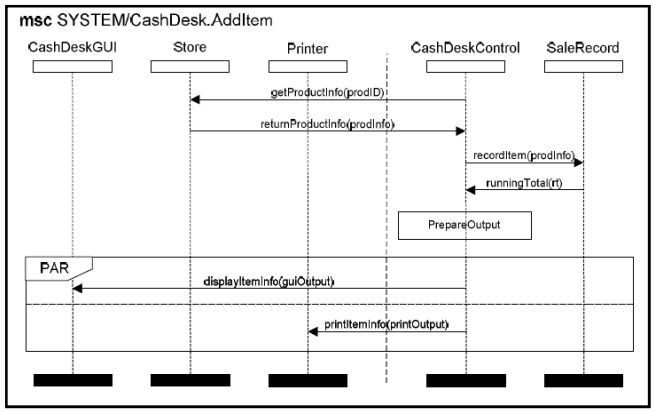


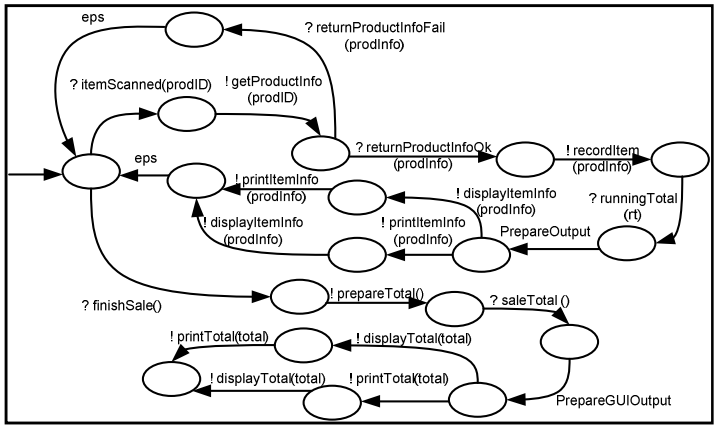












Logical Architecture Level

- Map Services to Logical Components
- Map Messages to Data Types
- Specify System Structure
- Transform Behavior Automata to FOCUS Timed State Transition Automata
- Complete the FOCUS specification

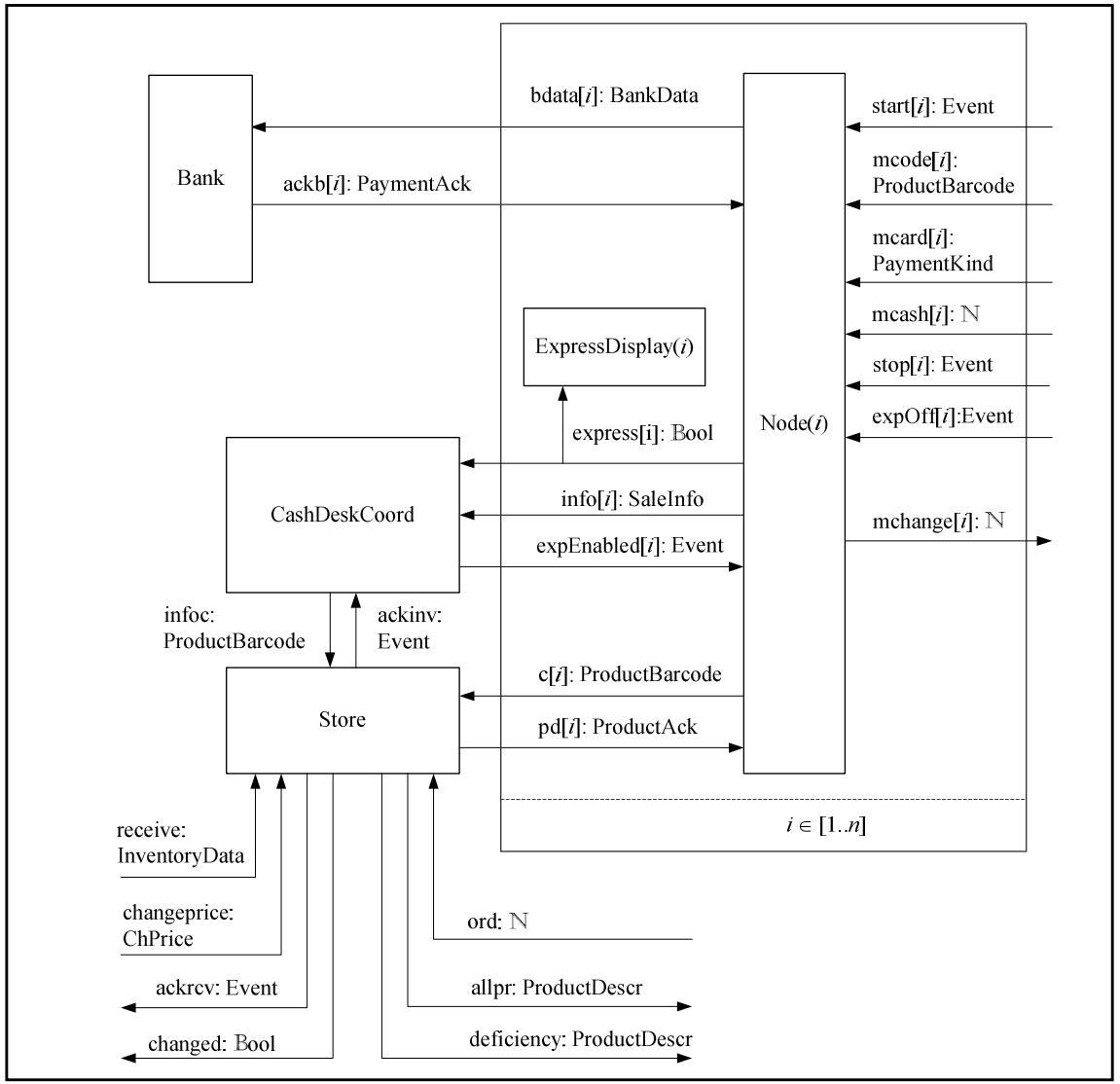
Mapping Services to Logical Components

Service	FOCUS Component	Type
CashDesk.ExpressChecker	*ExpressChecker*	component logic (*CashDeskCoord*)
CashDesk.Coordinator	*Coordinator*	component logic (*CashDeskCoord*)
CashDesk.SalesCache	*Coordinator*	local variable
CashDesk.CashDeskControl	*CashDeskControl*	component logic
CashDesk.ExpressStatus	*CashDeskControl*	local variable
CashDesk.SaleRecord	*CashDeskControl*	local variable

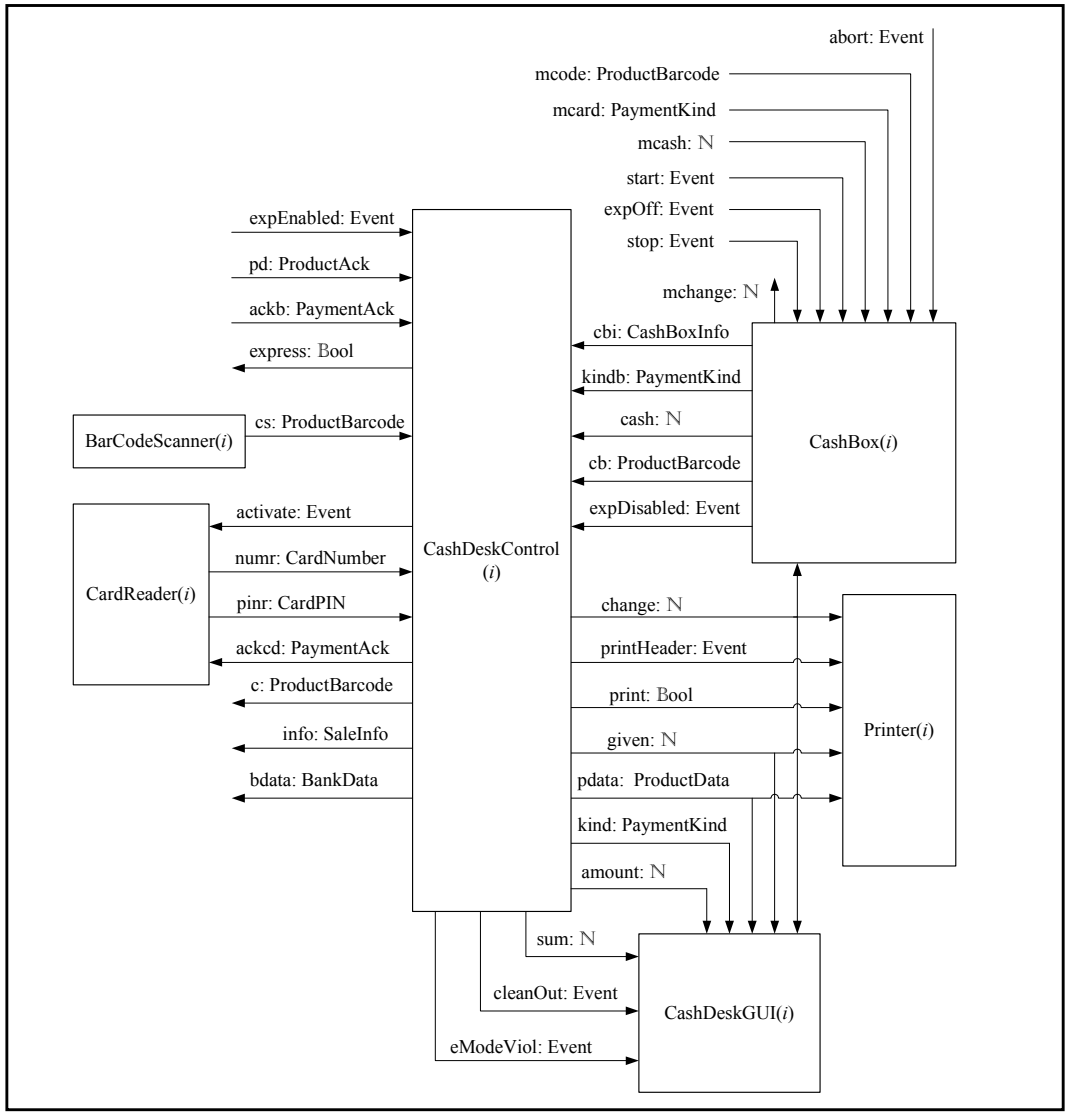
Mapping Messages to Data Structures

MSC Message	out	in	Data Type	Channels
startSale()	CashBox	CashDeskControl	CashBoxInfo	*cbi*
finishSale()	CashBox	CashDeskControl	CashBoxInfo	*cbi*
cashBoxClosed()	CashBox	CashDeskControls	CashBoxInfo	*cbi*
openCashBox()	CashBox	CashDeskControl	CashBoxInfo	*cbi*
cashPayment()	CashBox	CashDeskControl	PaymentKind	*kindb*
cardPayment()	CashBox	CashDeskControl	PaymentKind	*kindb*
itemScanned(...)	BarCodeScanner	CashDeskControl	ProductBarcode	*cs*
getProductInfo()	CashDesk	Inventory	ProductBarcode	*c*

CashDesk System Structure Specification



CashDesk Node Structure Specification

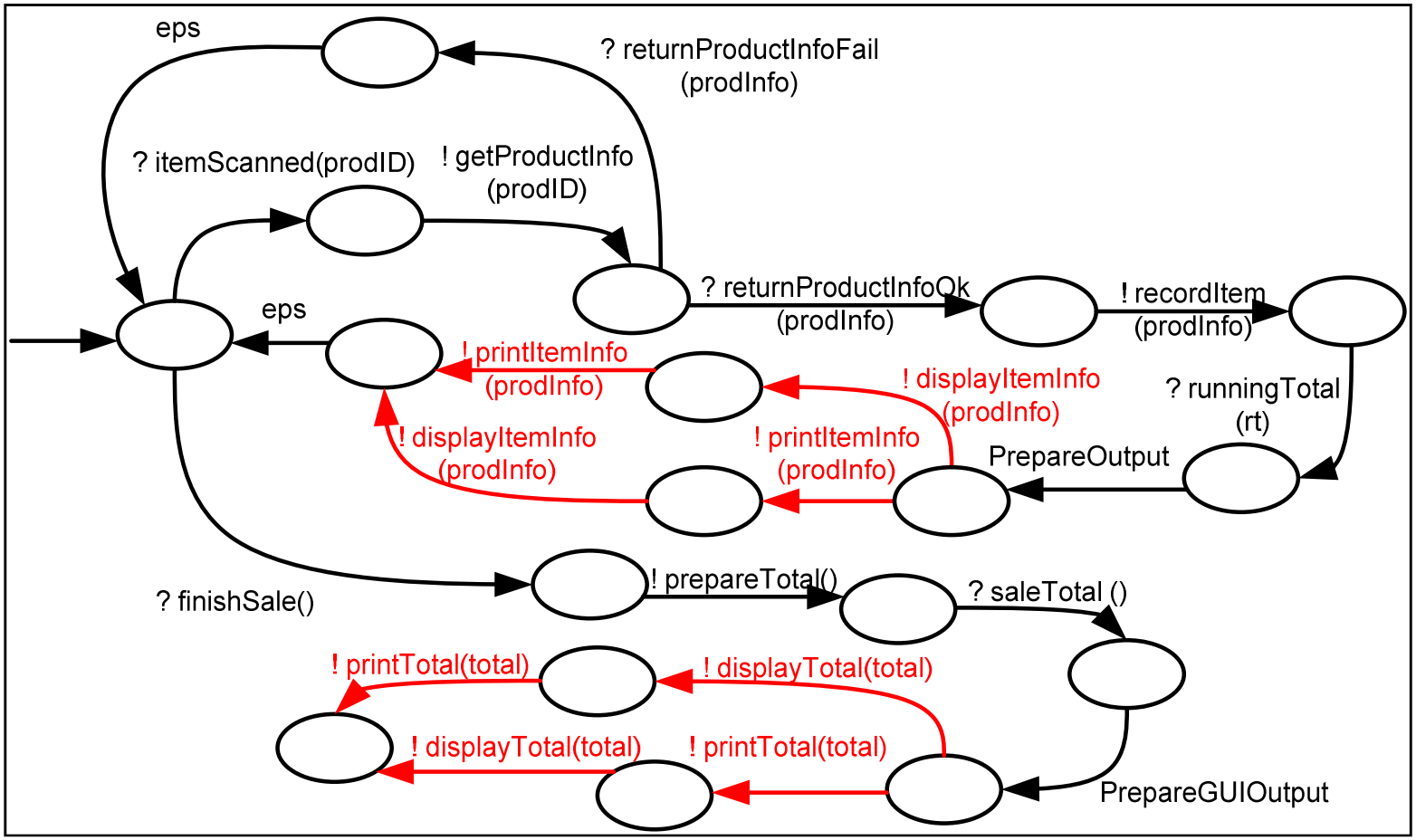


Transformation of Behavior Automata

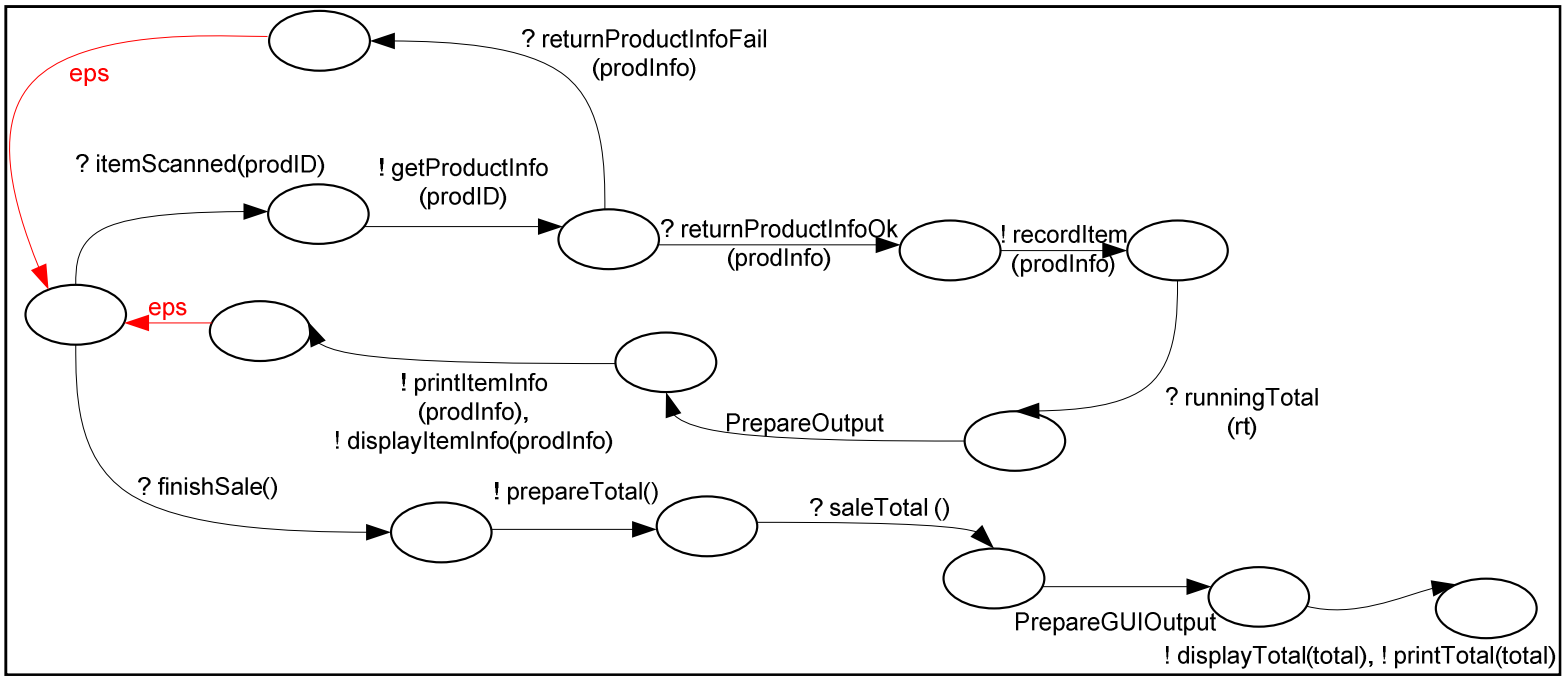
- Merge parallel output actions
- Remove epsilon transitions
- Merge local variable transformations
- Remove internal communication
- Merge general computation
- Transform messages into FOCUS syntax

- RESULT: FOCUS time state transition diagrams

Merge Parallel Output Actions

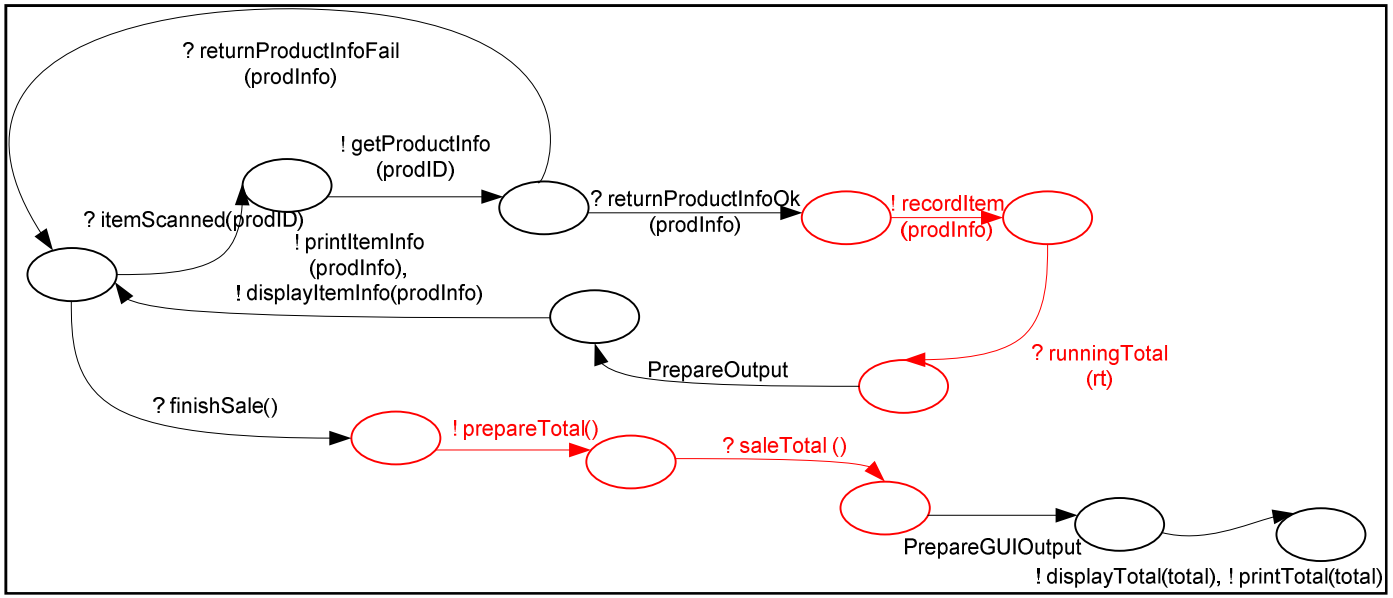


Remove Epsilon Transitions

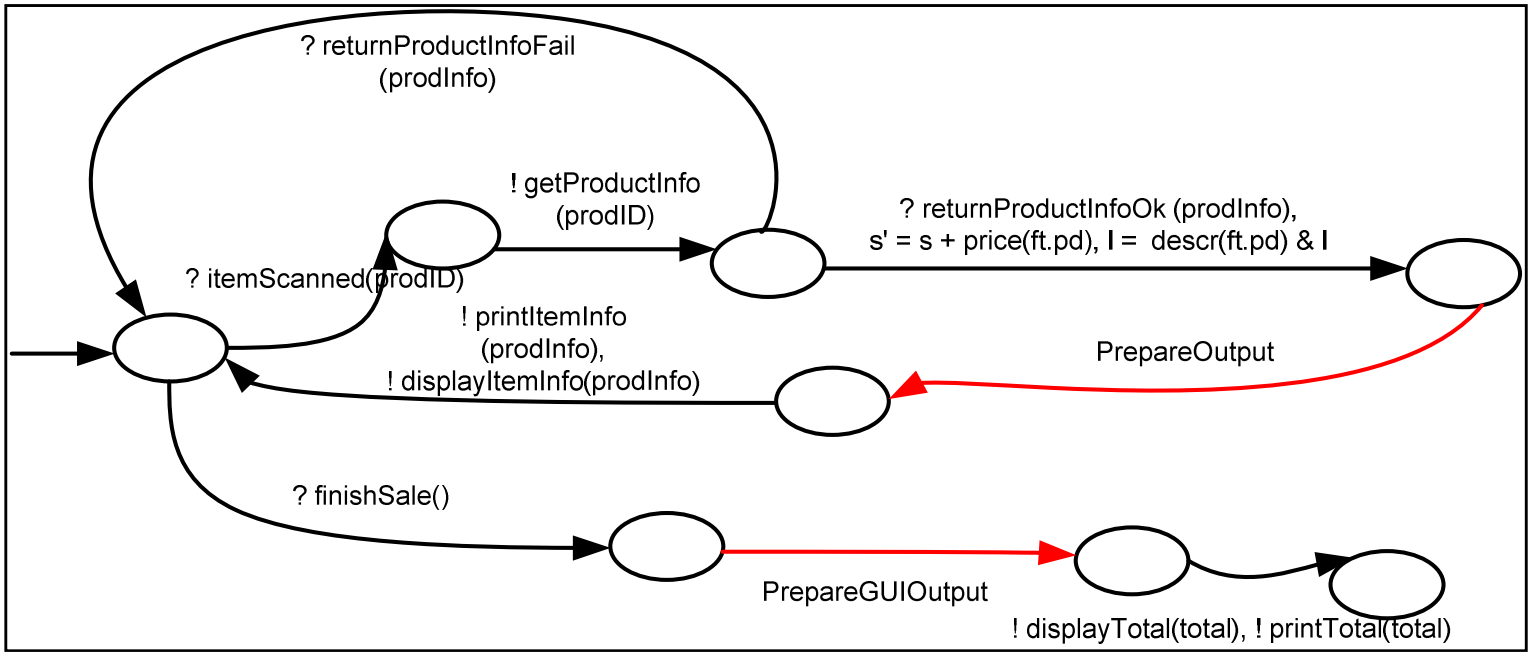


Merge Local Variable & Internal Comm.

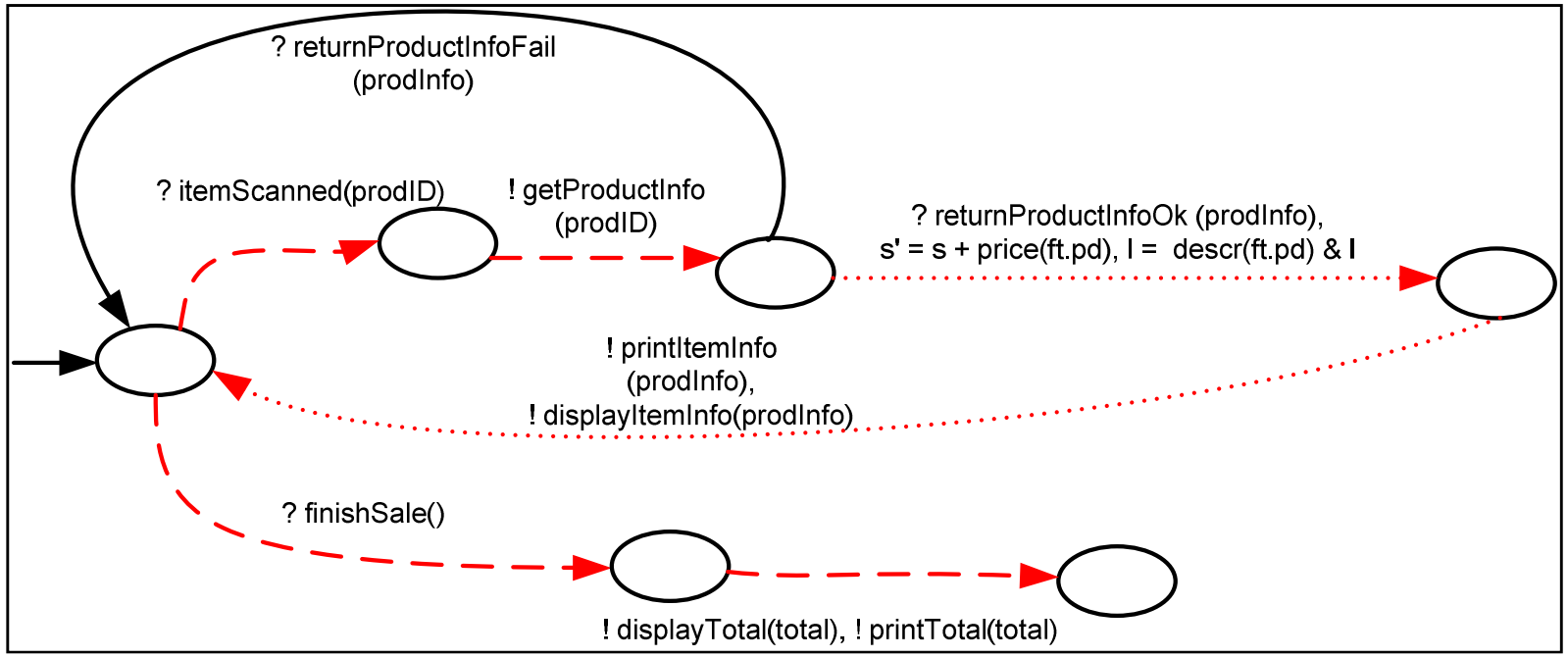
- Local Variables are updated
- Intra-component communication is removed



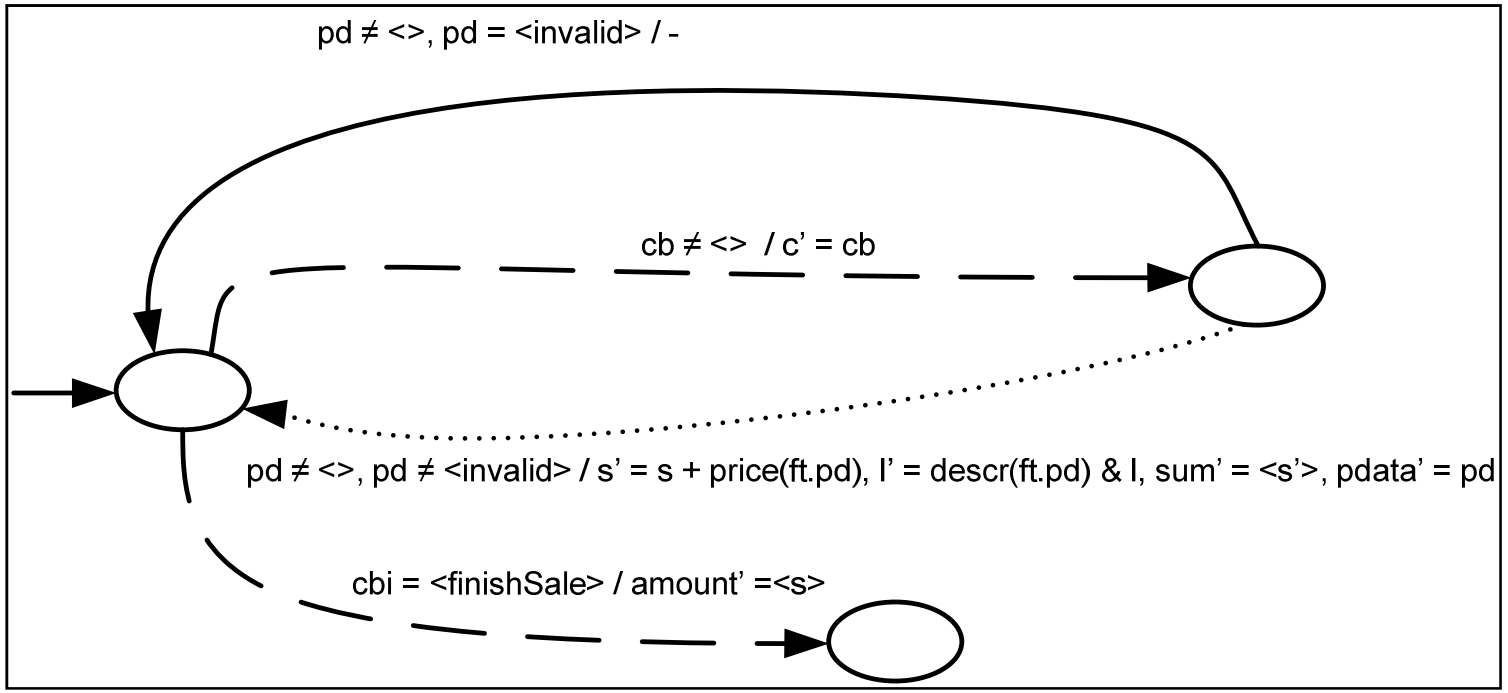
Merge General Computation



Transform to FOCUS Syntax



CashDeskControl Behavior Specification



Complete FOCUS Specification (I)

CashDeskControl — timed

in
$expEnabled, expDisabled : Event;\quad cbi : CashBoxInfo;$
$cash : \mathbb{N};\quad cb, cs : ProductBarcode;\quad pd : ProductAck;\quad ackb : PaymentAck;$
$numr : CardNumber;\quad pinr : CardPIN;\quad kindb : PaymentKind;$

out
$c : ProductBarcode;\quad info : SaleInfo;\quad bdata : BankData;\quad express : \mathbb{B}ool;$
$amount, sum, given, change : \mathbb{N};\quad ackcd : PaymentAck;$
$print : \mathbb{B}ool;\quad printHeader, activate, cleanOut, eModeViol : Event;$
$pdata : ProductData;\quad kind : PaymentKind;$

local
$st \in Status;\quad l \in ProductDescr^*;\quad cnum \in CardNumber;$
$mode, modecash \in \mathbb{B}ool;\ s \in \mathbb{N}$

univ
$x, y : Event^*;\ xcbi \in CashBoxInfo^*;\quad xk \in PaymentKind^*;\quad xcash \in \mathbb{N}^*$
$xcb, xcs \in ProductBarcode^*;\quad xpd \in ProductAck^*;\quad xackb \in PaymentAck^*;$
$xnumr \in CardNumber^*;\quad xpinr \in CardPIN^*;$

Complete FOCUS Specification (II)

init $st = init;\ \ l = \langle\rangle;\ \ mode = \text{false};\ \ modecash = \text{false};\ \ s = 0;$

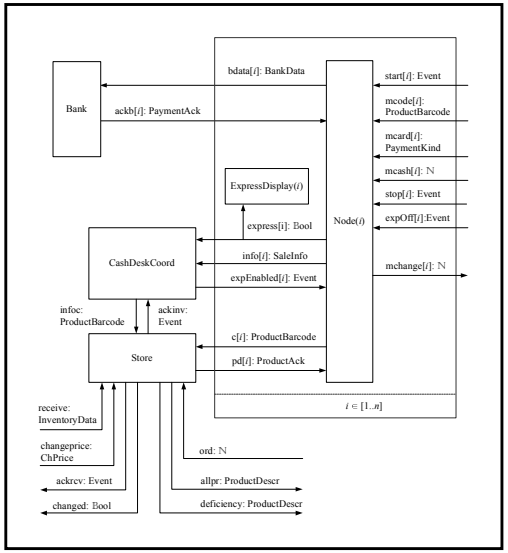
asm disjoint(cb, cs)
$\text{msg}_1(cbi) \wedge \text{msg}_1(kindb) \wedge \text{msg}_1(cash) \wedge \text{msg}_1(cb)$
$\text{msg}_1(cs) \wedge \text{msg}_1(pd) \wedge \text{msg}_1(ackb) \wedge \text{msg}_1(numr) \wedge \text{msg}_1(pinr)$

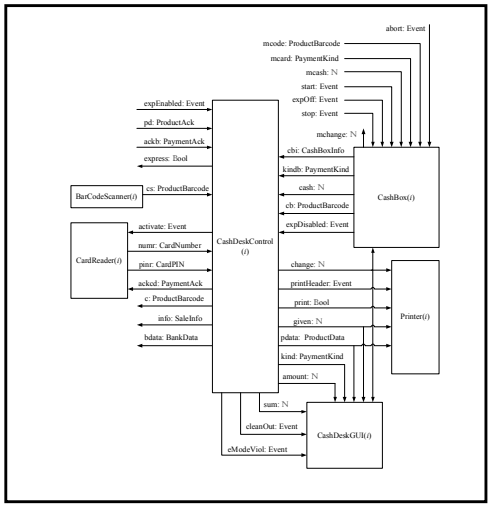
gar $\text{ti}(c, 0) = \langle\rangle \wedge \text{ti}(info, 0) = \langle\rangle \wedge \text{ti}(bdata, 0) = \langle\rangle$
$\wedge\ \text{ti}(cleanOut, t) = \langle\rangle \wedge \text{ti}(eModeViol, t) = \langle\rangle$
$\wedge\ \text{ti}(express, 0) = \langle\rangle \wedge \text{ti}(amount, 0) = \langle\rangle \wedge \text{ti}(sum, 0) = \langle\rangle$
$\wedge \text{ti}(given, 0) = \langle\rangle$
$\wedge\ \text{ti}(change, 0) = \langle\rangle \wedge \text{ti}(ackcd, 0) = \langle\rangle \wedge \text{ti}(print, 0) = \langle\rangle$
$\wedge \text{ti}(printHeader, 0) = \langle\rangle$
$\wedge\ \text{ti}(activate, 0) = \langle\rangle \wedge \text{ti}(kind, 0) = \langle\rangle \wedge \text{ti}(pdata, 0) = \langle\rangle$

tiTable $ExpressCheckerTable$

tiAutomat$CashDeskControlAutomat$

Logical Architecture Level Results





CashDeskControl — timed

in
$expEnabled, expDisabled : Event;\quad cbi : CashBoxInfo;$
$cash : \mathbb{N};\quad cb, cs : ProductBarcode;\quad pd : ProductAck;\quad ackb : PaymentAck;$
$numr : CardNumber;\quad pinr : CardPIN;\quad kindb : PaymentKind;$

out
$c : ProductBarcode;\quad info : SaleInfo;\quad bdata : BankData;\quad express : \mathbb{B}ool;$
$amount, sum, given, change : \mathbb{N};\quad ackcd : PaymentAck;$
$print : \mathbb{B}ool;\quad printHeader, activate, cleanOut, eModeViol : Event;$
$pdata : ProductData;\quad kind : PaymentKind;$

local
$st \in Status;\quad l \in ProductDescr^*;\quad cnum \in CardNumber;$
$mode, modecash \in \mathbb{B}ool;\ s \in \mathbb{N}$

univ
$x, y : Event^*;\ xcbi \in CashBoxInfo^*;\quad xk \in PaymentKind^*;\quad xcash \in \mathbb{N}^*$
$xcb, xcs \in ProductBarcode^*;\quad xpd \in ProductAck^*;\quad xackb \in PaymentAck^*;$
$xnumr \in CardNumber^*;\quad xpinr \in CardPIN^*;$

asm
$\mathrm{disjoint}(cb, cs)$
$\mathrm{msg}_1(cbi) \wedge \mathrm{msg}_1(kindb) \wedge \mathrm{msg}_1(cash) \wedge \mathrm{msg}_1(cb)$
$\mathrm{msg}_1(cs) \wedge \mathrm{msg}_1(pd) \wedge \mathrm{msg}_1(ackb) \wedge \mathrm{msg}_1(numr) \wedge \mathrm{msg}_1(pinr)$

gar
$\mathrm{ti}(c, 0) = \langle\rangle \wedge \mathrm{ti}(info, 0) = \langle\rangle \wedge \mathrm{ti}(bdata, 0) = \langle\rangle$
$\wedge\ \mathrm{ti}(cleanOut, t) = \langle\rangle \wedge \mathrm{ti}(eModeViol, t) = \langle\rangle$
$\wedge\ \mathrm{ti}(express, 0) = \langle\rangle \wedge \mathrm{ti}(amount, 0) = \langle\rangle \wedge \mathrm{ti}(sum, 0) = \langle\rangle$
$\wedge \mathrm{ti}(given, 0) = \langle\rangle$
$\wedge\ \mathrm{ti}(change, 0) = \langle\rangle \wedge \mathrm{ti}(ackcd, 0) = \langle\rangle \wedge \mathrm{ti}(print, 0) = \langle\rangle$
$\wedge \mathrm{ti}(printHeader, 0) = \langle\rangle$
$\wedge\ \mathrm{ti}(activate, 0) = \langle\rangle \wedge \mathrm{ti}(kind, 0) = \langle\rangle \wedge \mathrm{ti}(pdata, 0) = \langle\rangle$

tiTable *ExpressCheckerTable*

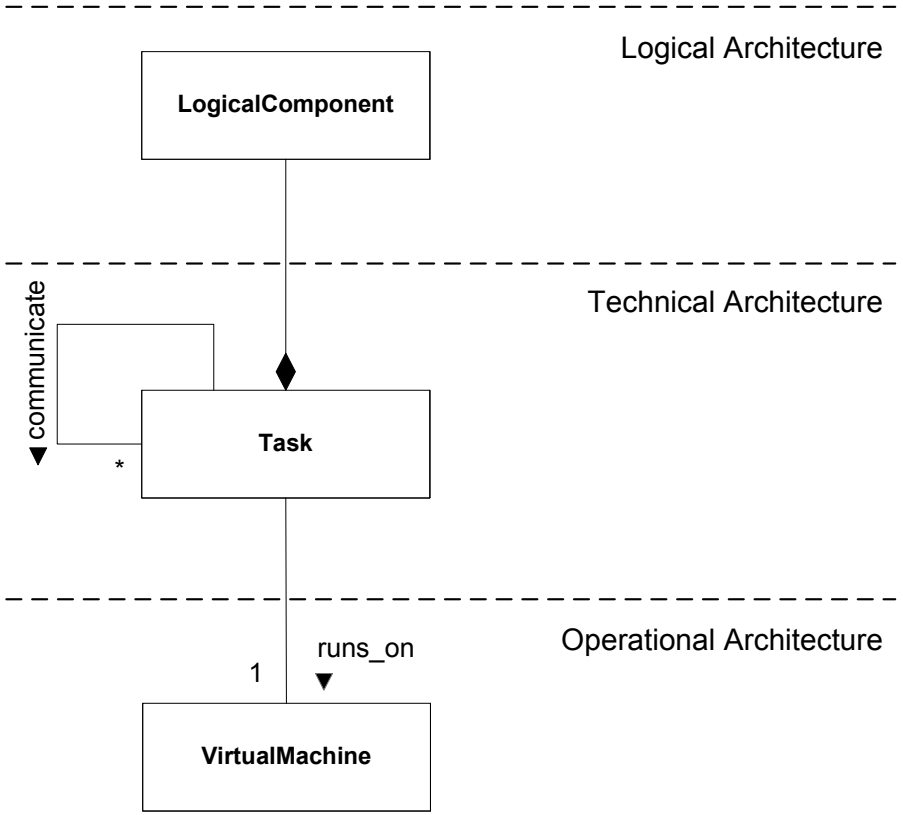
tiAutomat *CashDeskControlAutomat*

Deployment Level

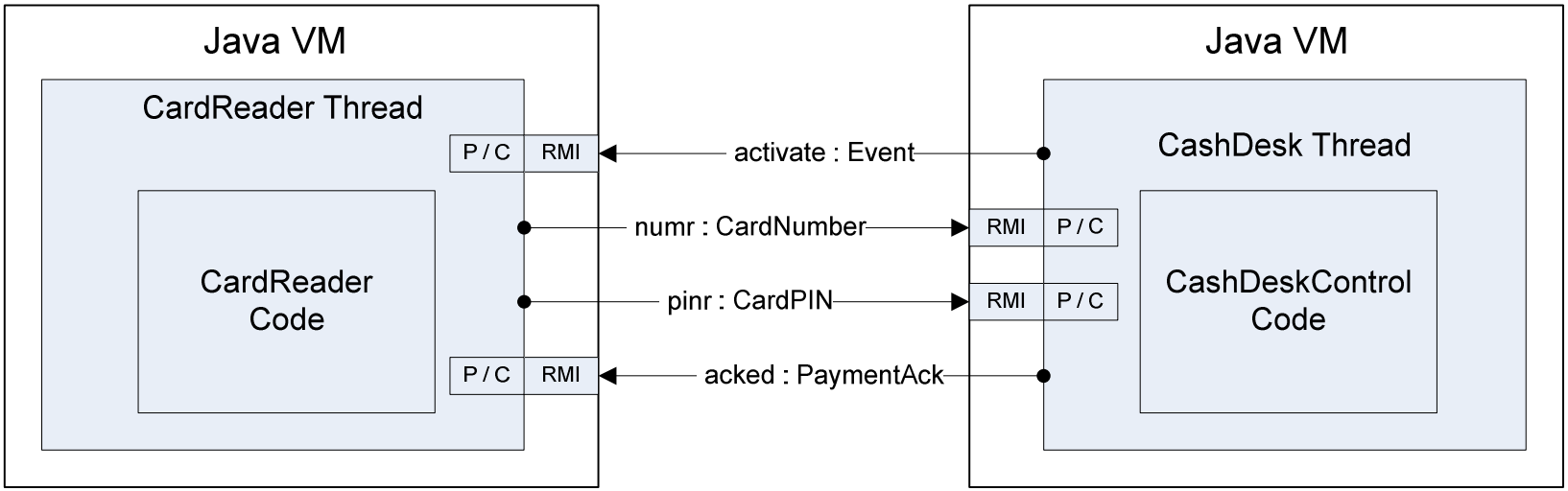
- Technical Architecture
  - Components are arbitrarily clustered into Tasks
  - Tasks form executable objects implementing the behavior
- Operational Architecture
  - Deployment Infrastructure
    - Thread
    - Remote Method Invocation Facility
  - Execution Environment / Target Platform
    - Java Virtual Machines
- TA + OA + External IFC = Executable System

Deployment Methodology

- Embed every logical component in a Task
- Embed entities of the technical architecture into entities of the operational architecture
- Most code is generated
- External interfaces connected manually

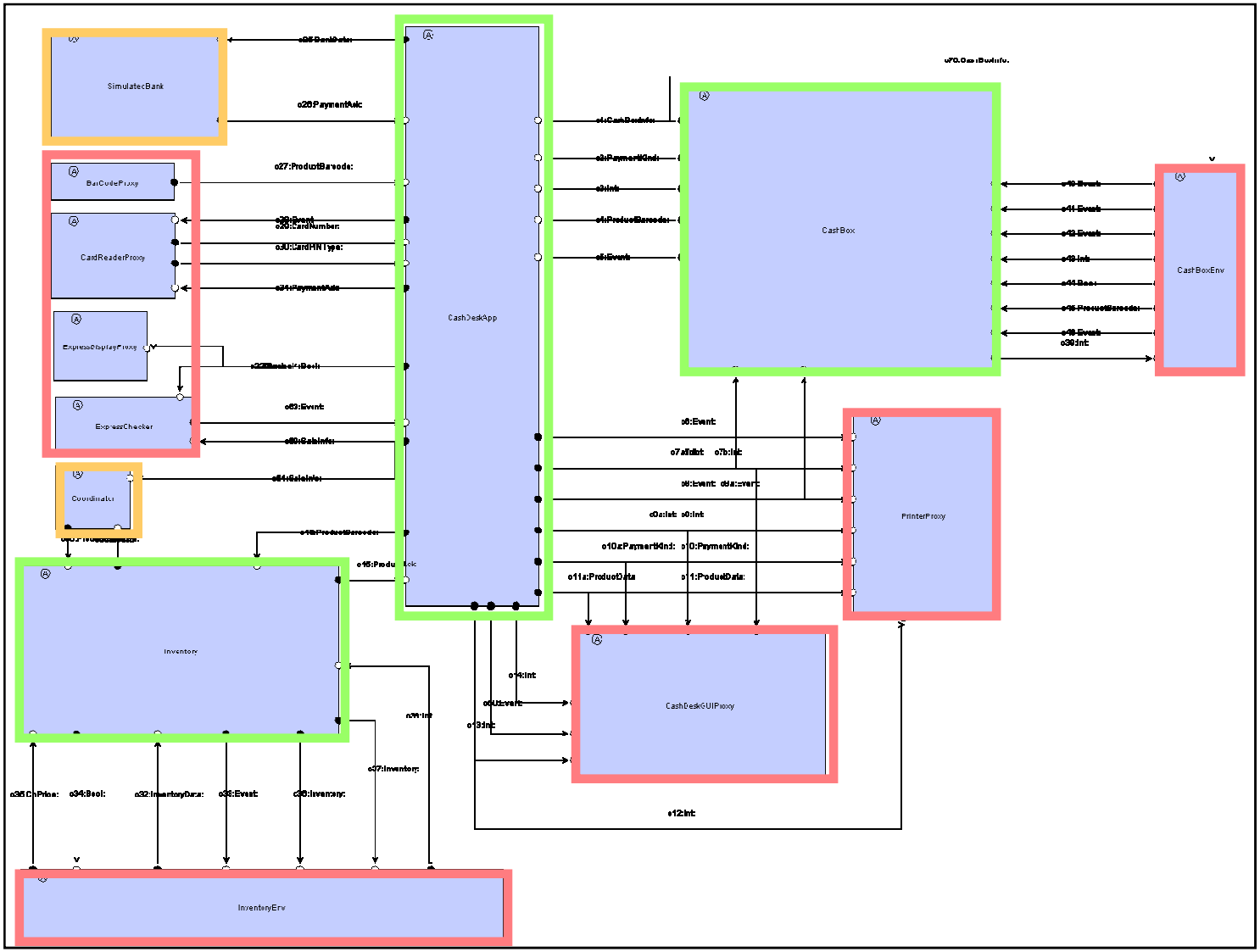


Deployment Example



- RMI calls synchronized by Producer / Consumer
- Strong Causality ensures Dead-lock freedom
- Synchronous Message Exchange + NoVal Messages = Asynchronous Communication

AutoFOCUS 2 Model: Mock-up Prototype



Handwritten Implementation: Mock-up PT

- Manual part override dummy automata

```
private JTextField field;
private Object lock = new Object();
private Integer code = null;

public void do_step() {
    synchronized (lock) {
        if(code!=null) {
            cs.setValProductBarcode(ProductBarcode.valueOf("Code("+code.toString()+")"));
            code = null;
        }
        else
            cs.setNoVal();
}}
public void setCode(int c) {
    synchronized(lock) {
        code = new Integer(c);
}}
public void actionPerformed(ActionEvent arg0) {
    try {
        int x = Integer.parseInt(field.getText());
        this.setCode(x);
        field.setText("");
    } catch (NumberFormatException ex) {}
}
```

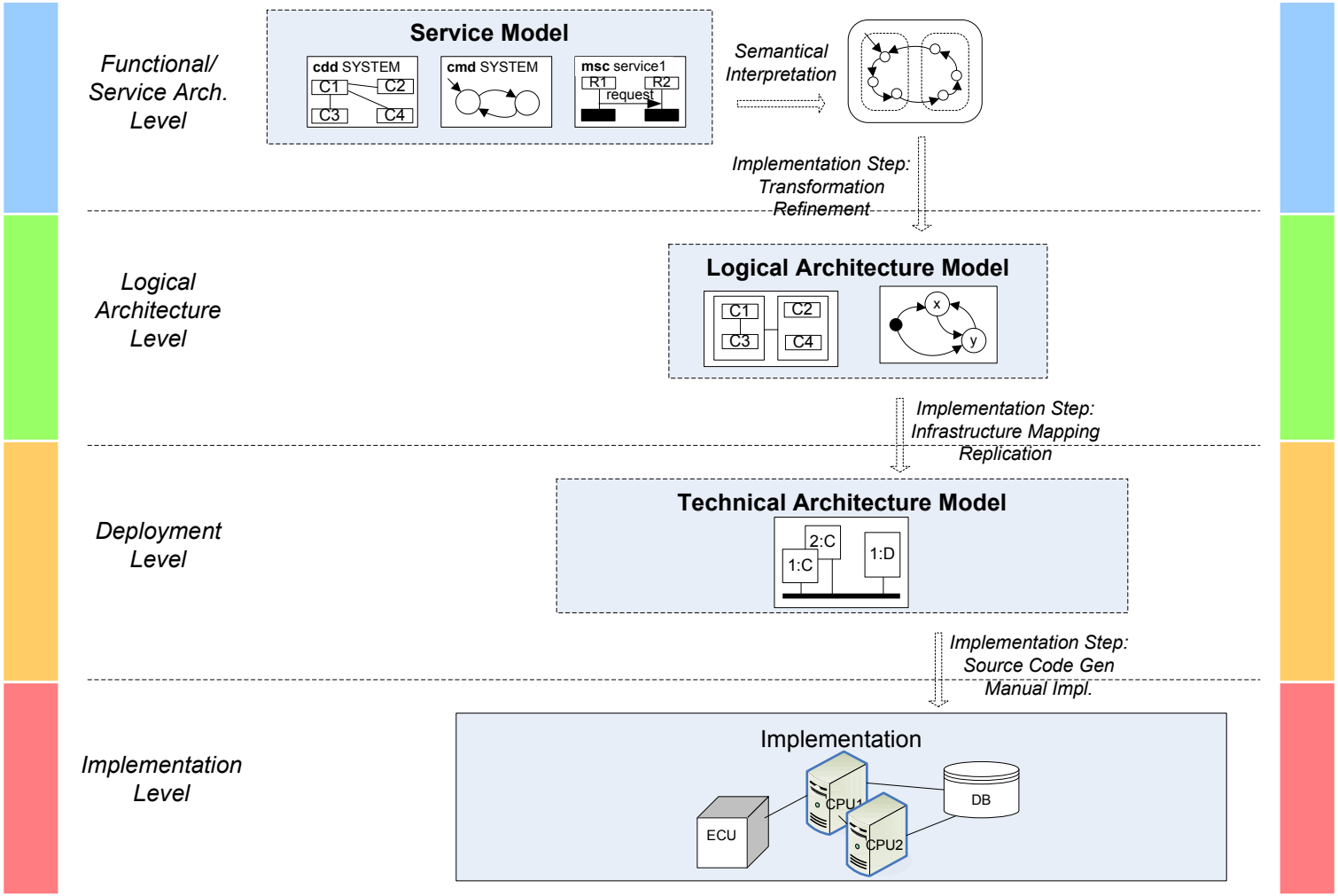
Part III: Conclusion and Experiences

Summary

Lessons Learned

TECHNISCHE UNIVERSITÄT MÜNCHEN

Summary


Functional/ Service Arch. Level
Service Model
cdd SYSTEM
C1
C2
C3
C4
cmd SYSTEM
msc service1
R1
R2
request
Semantical Interpretation
Implementation Step: Transformation Refinement
Logical Architecture Level
Logical Architecture Model
C1
C2
C3
C4
x
y
Implementation Step: Infrastructure Mapping Replication
Deployment Level
Technical Architecture Model
2:C
1:C
1:D
Implementation Step: Source Code Gen Manual Impl.
Implementation Level
Implementation
ECU
CPU1
CPU2
DB

Lessons Learned

- Instantiation changes system structure in Deployment
  - Connecting N cashdesks with the inventory changes the inventory behavior
  - Using a merger / bus component changes delay in the communication from cashdesk to inventory
- Our methodology works for distributed teams
  - Service Level Team analyzed requirements
  - Logical Level Team used service architecture specification
  - Deployment Team used logical architecture specification

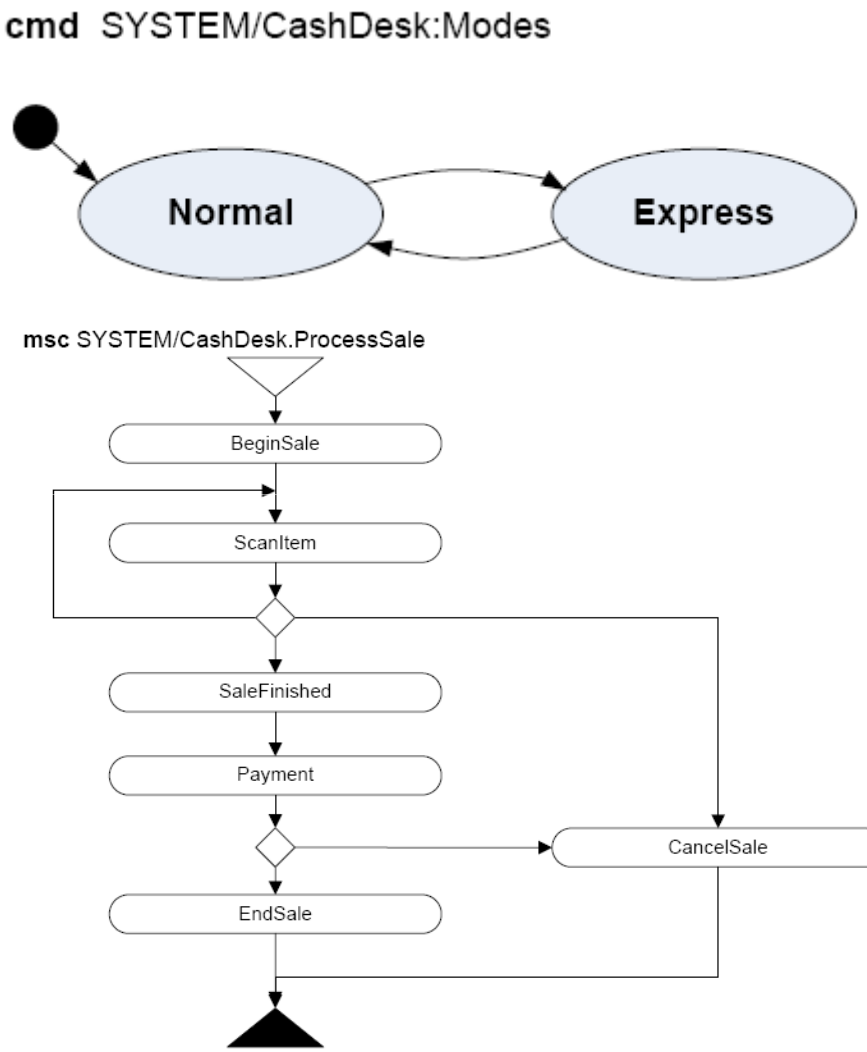
Part IV: Demonstration

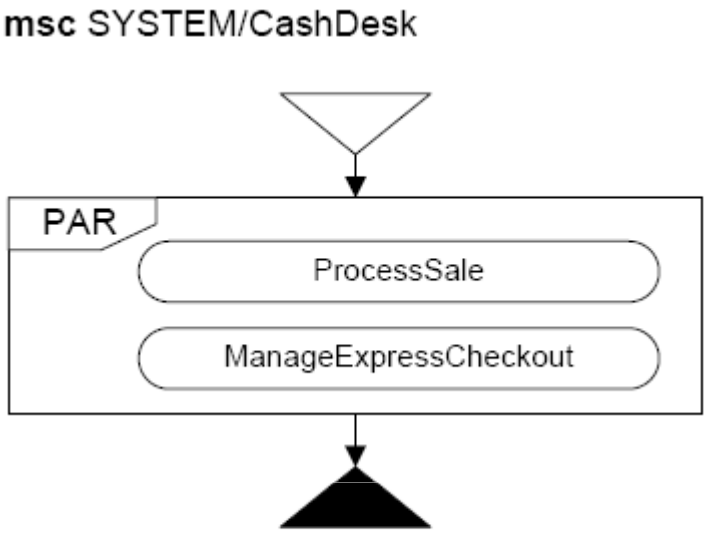
AutoFOCUS 2 Model

System in Action

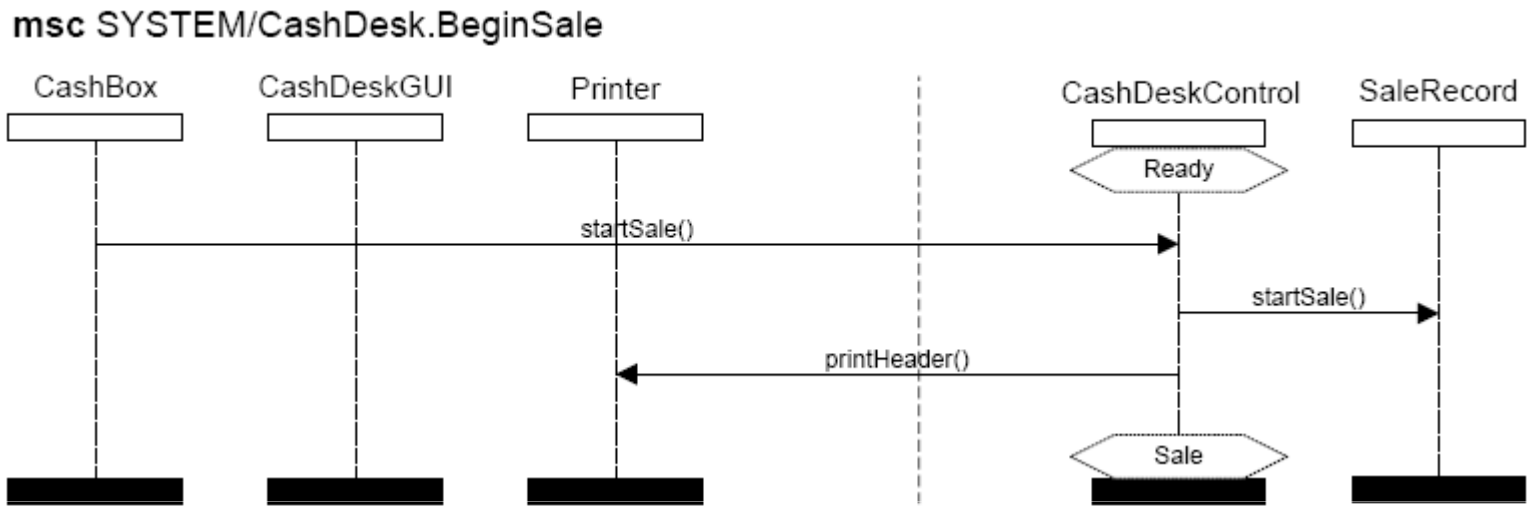
Backup

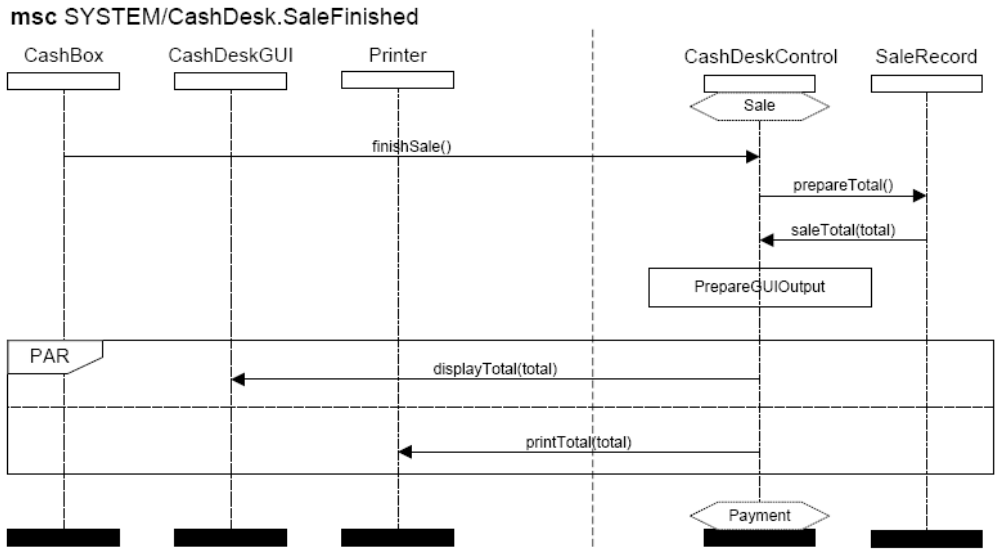
Detailed Service Specification I



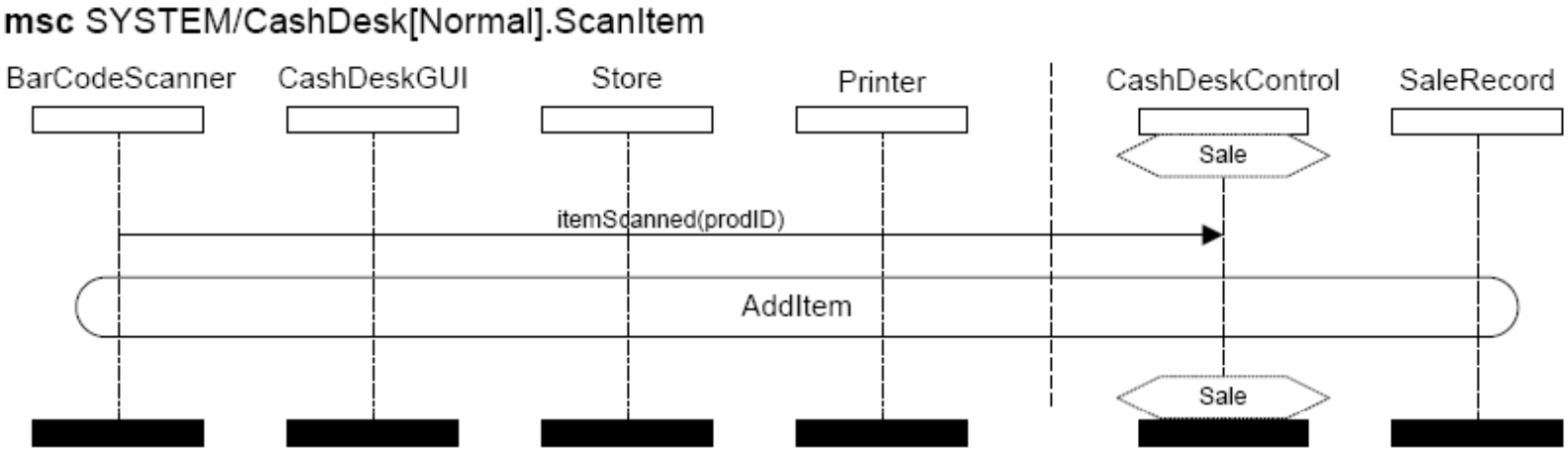


Detailed Service Specification II

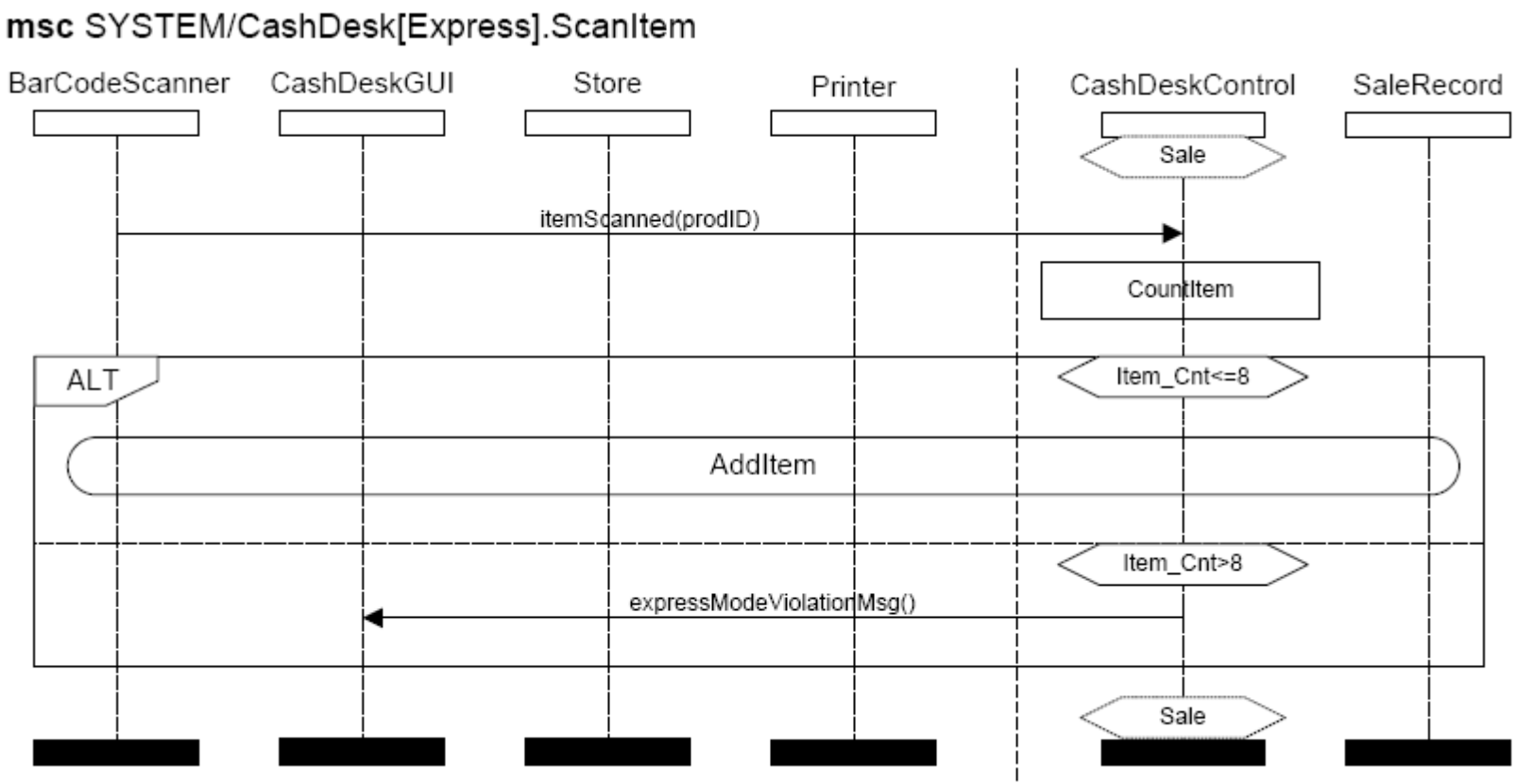




Detailed Service Specification III



Detailed Service Specification IV



Parallel reaction problem

$\forall i \in [1..n] : \quad \forall t \in \mathbb{N} :$
if ti$(c_i, t) \neq \langle\rangle$
then ti$(pd_i, t+1) = takeProductData(\text{ft.ti}(c_i, t),\ invtable)$
else ti$(pd_i, t+1) = \langle\rangle$ fi

$\forall t \in \mathbb{N} :$
if ti$(infoc, t) \neq \langle\rangle$
then ti$(ackinv, t+$
else ti$(ackinv, t+$

$\forall t \in \mathbb{N} :$
if ti$(changeprice,$
then if $\exists x \in invt$
then ti$(chan$
else ti$(chang$
else ti$(changed, t$