UNIVERSITÀ DEGLI STUDI DI BERGAMO
Dipartimento di Ingegneria Gestionale, dell'Informazione e della Produzione

CoCoME – The Common Component Modeling Example

PROGETTAZIONE, ALGORITMI E COMPUTABILITÀ (38090-MOD1)

Corso di laurea Magistrale in Ingegneria Informatica

RELATORE
Prof.ssa Patrizia Scandurra

SEDE
DIGIP

DATA
04-10-2021

References and outline

http://www.cocome.org/

See document:
http://www.cocome.org/downloads/documentation/cocome.pdf

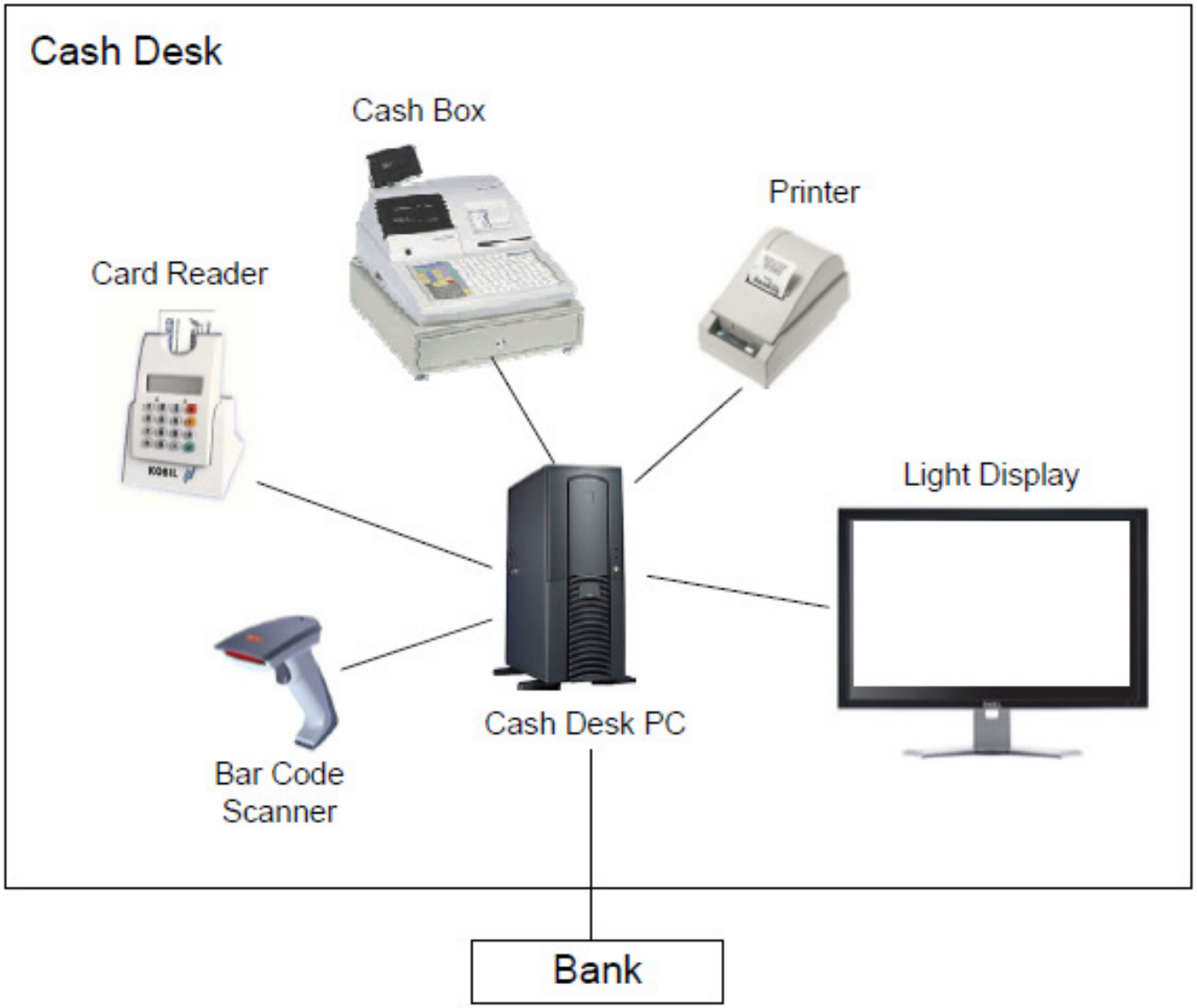
- Introduction and system overview
- Functional Requirements and Use Cases
- Software architecture design
  - Structural view
  - Deployment view
- Implementation aspects

CoCoME: Introduction and System Overview (Sect. 1.1)

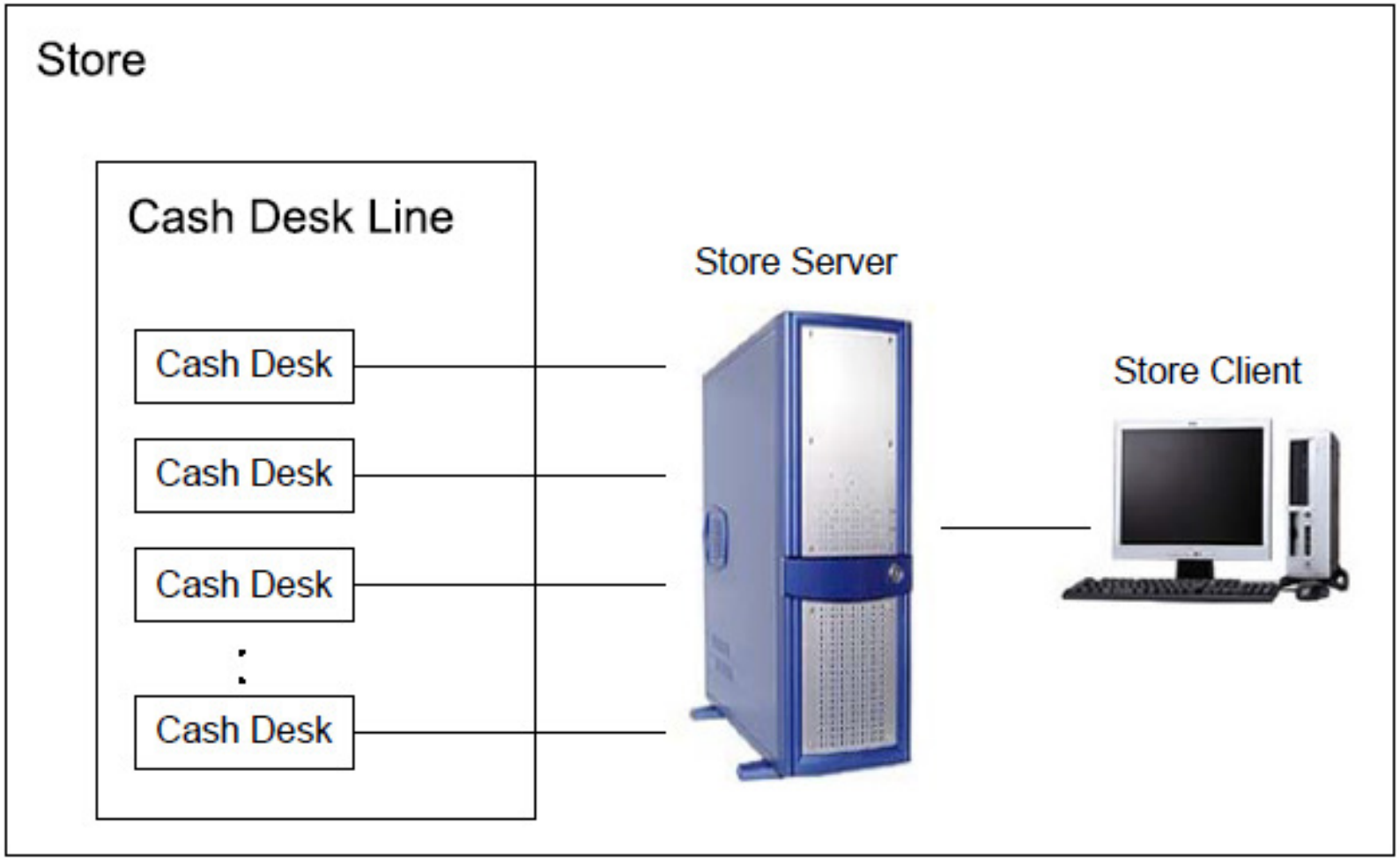
Introduction

- The example describes a *Trading System* as it can be observed in a supermarket handling sales.
- This includes:
  - the processes at a single Cash Desk like scanning products using a Bar Code Scanner or paying by credit card or by cash, as well as
  - administrative tasks like ordering of running out products or generating reports.
- Furthermore, a cash desk can switch into an *express checkout mode* denoted by a Light Display to allow:
  - only costumers with a few goods and also
  - only cash payment to speed up the clearing.

Hardware components of a single Cash Desk

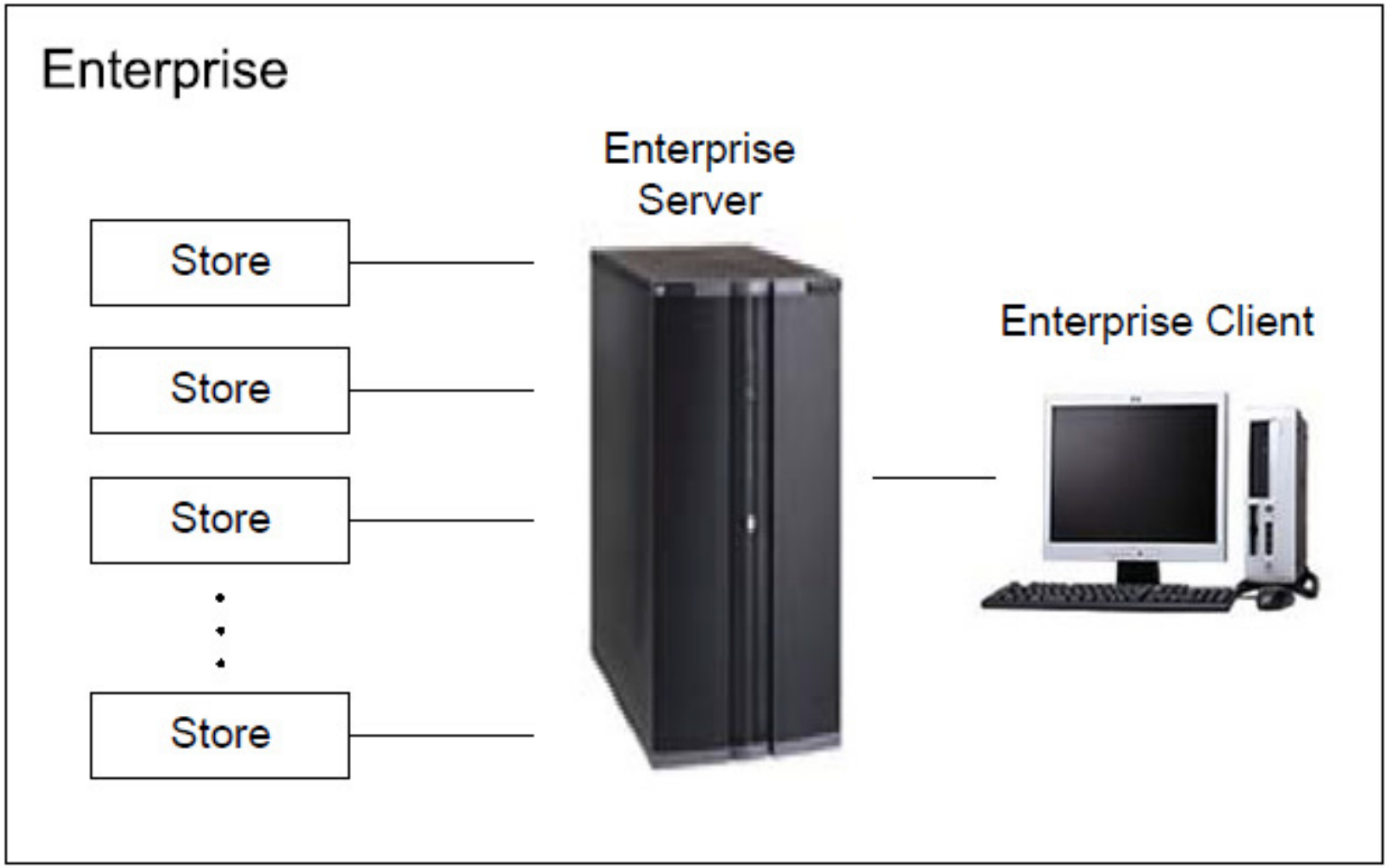


Entities in a single store



- The Store Server also holds the Inventory of the corresponding Store.
- The Store Client is used by the manager of the Store to view reports, order products or to chance the sales prices of goods.

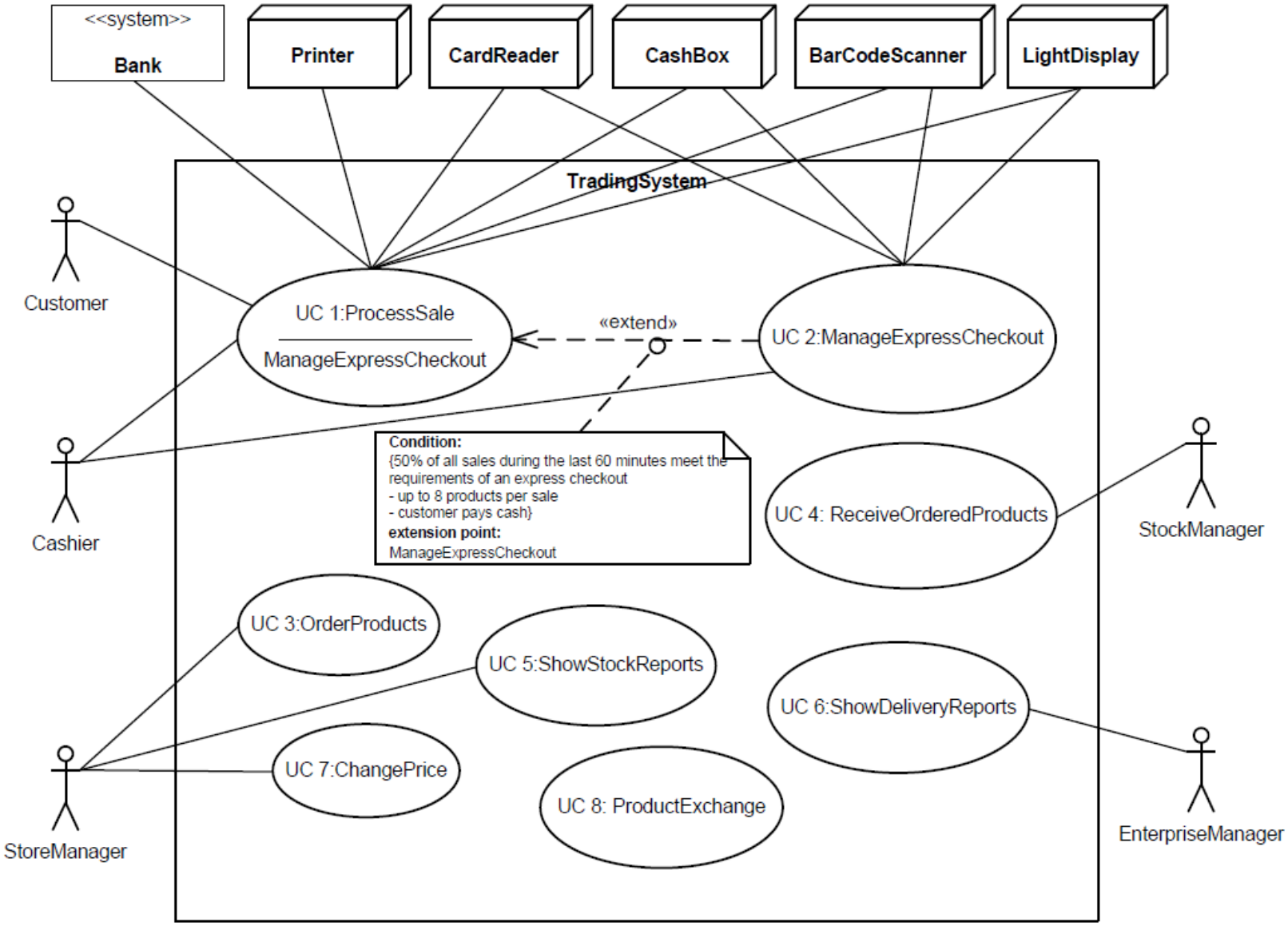
A set of Stores organized in an Enterprise



CoCoME: Functional Requirements and Use Cases (Sect. 1.2)

An overview of all use cases

The codes in the squared brackets refer to *extra-functional properties*.



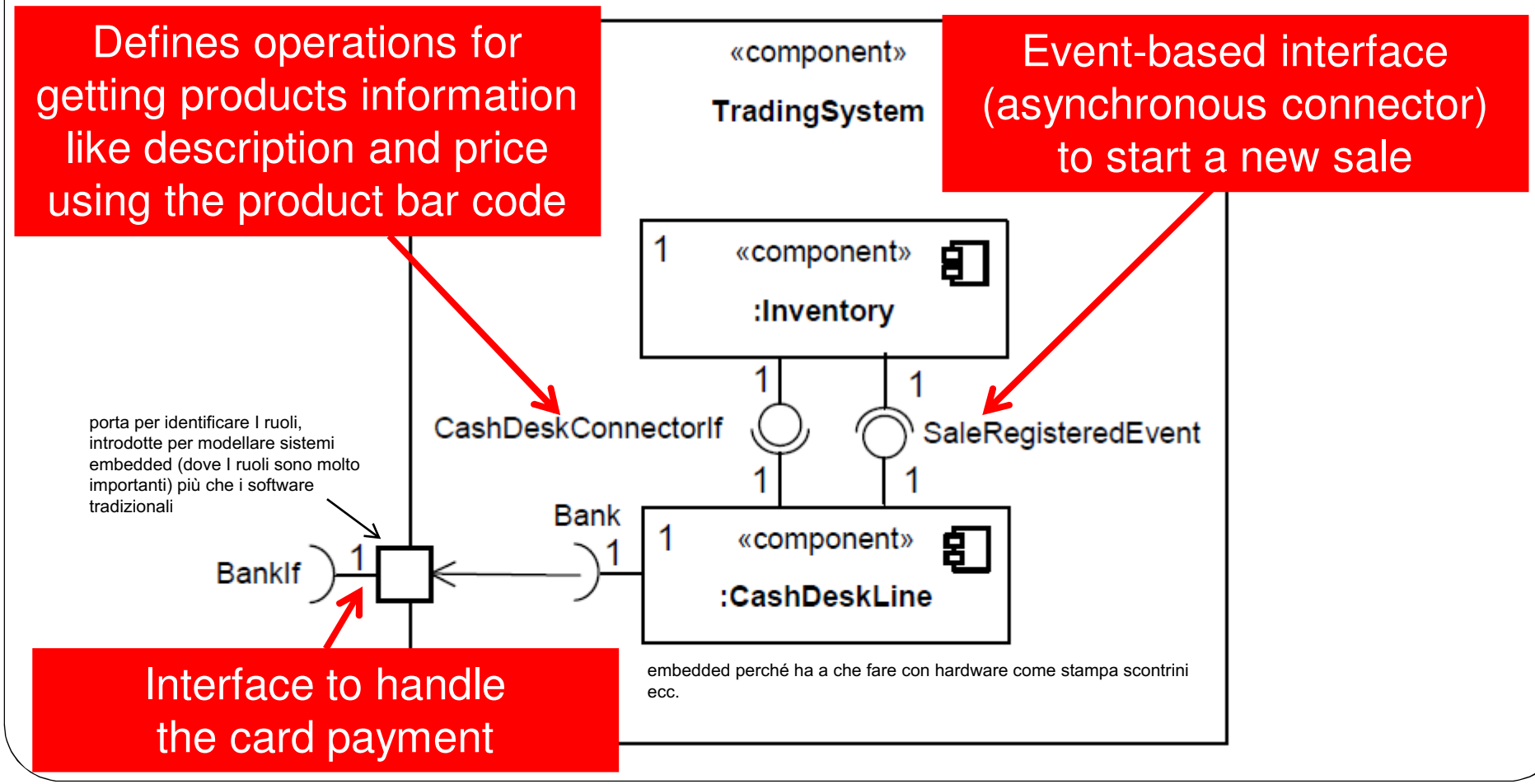
Use case descriptions

- Each use case is described using a uniform template which includes:
  - a brief description of the use case itself,
  - the standard process flow and its alternatives,
  - Information like preconditions, postconditions and the trigger of the use cases.
- See Sec. 1.2 of the pdf document.
  - See, for example, *UC 1:ProcessSale* and *UC 2:ManageExpressCheckout*

SoCoME: Software architecture design
(Sect. 1.4)

Structural view of the Trading System

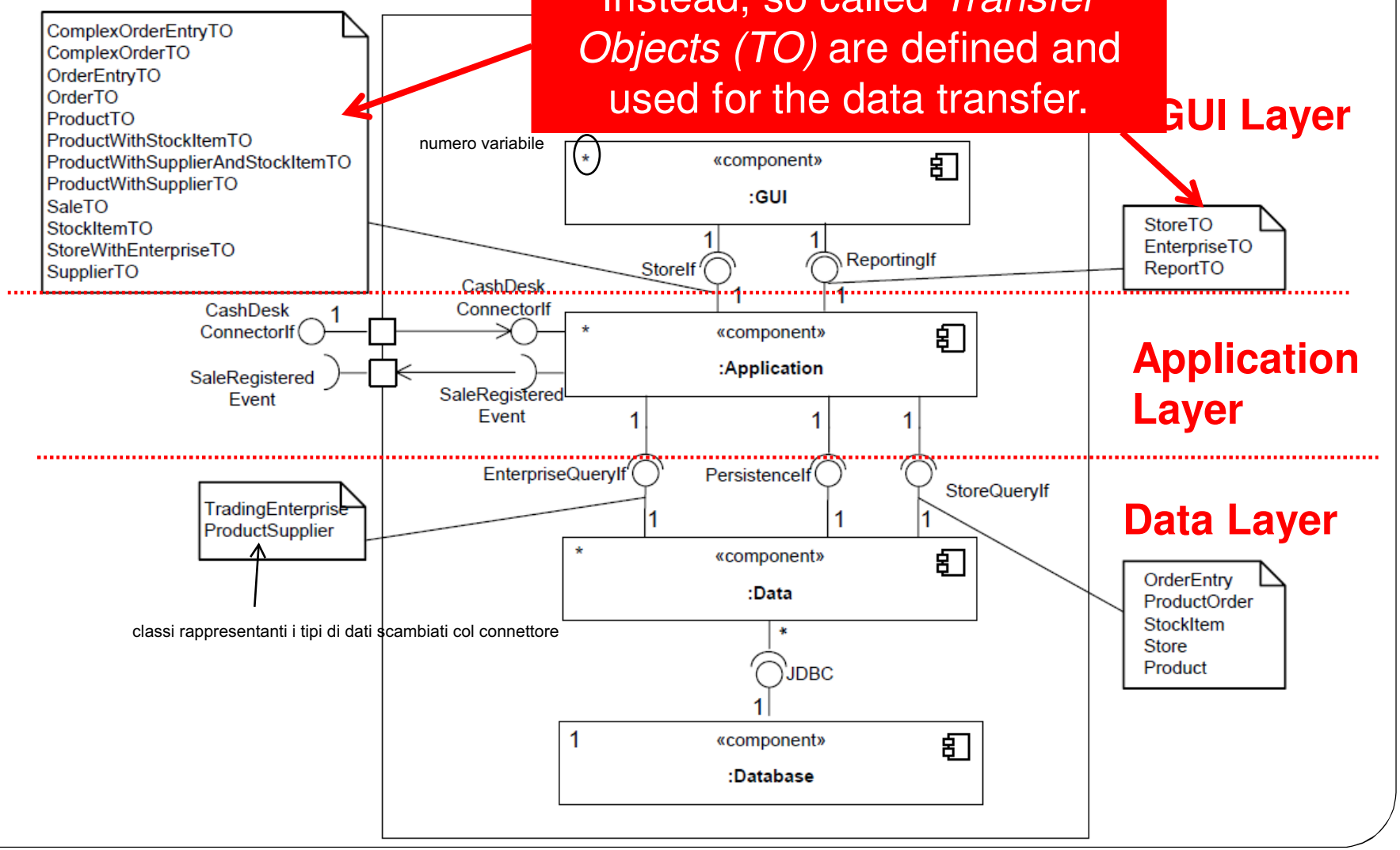
- Hierarchical design, UML component diagrams with multiplicities and ports
- The *information system* is represented by the component **Inventory**, while the component **CashDeskLine** represents the *embedded system*



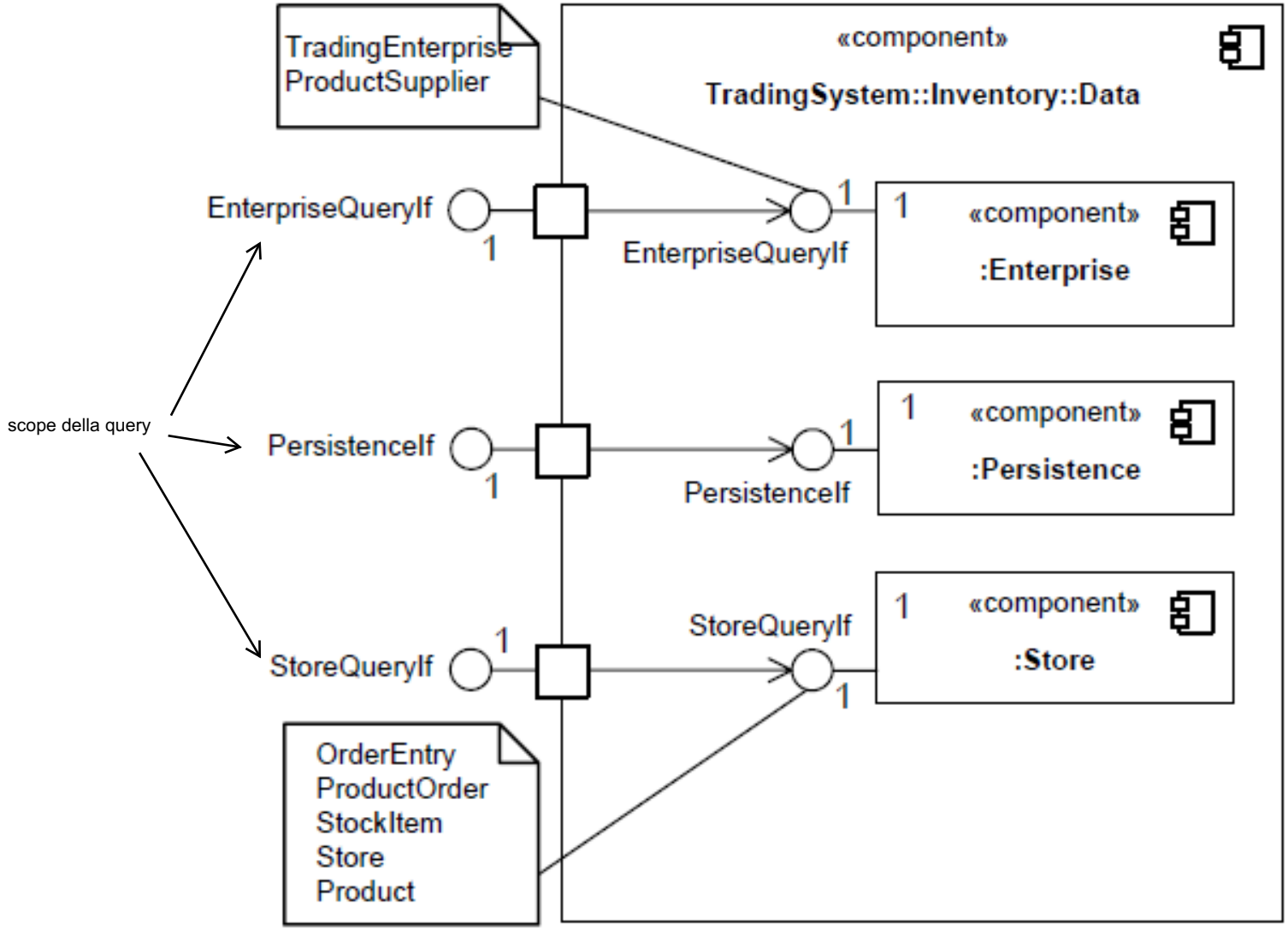
The inner structure of the component Inventory

- Organized as a *three-layer architecture*: *GUI*, *Application*, and *Data*, and completed by a component *I*

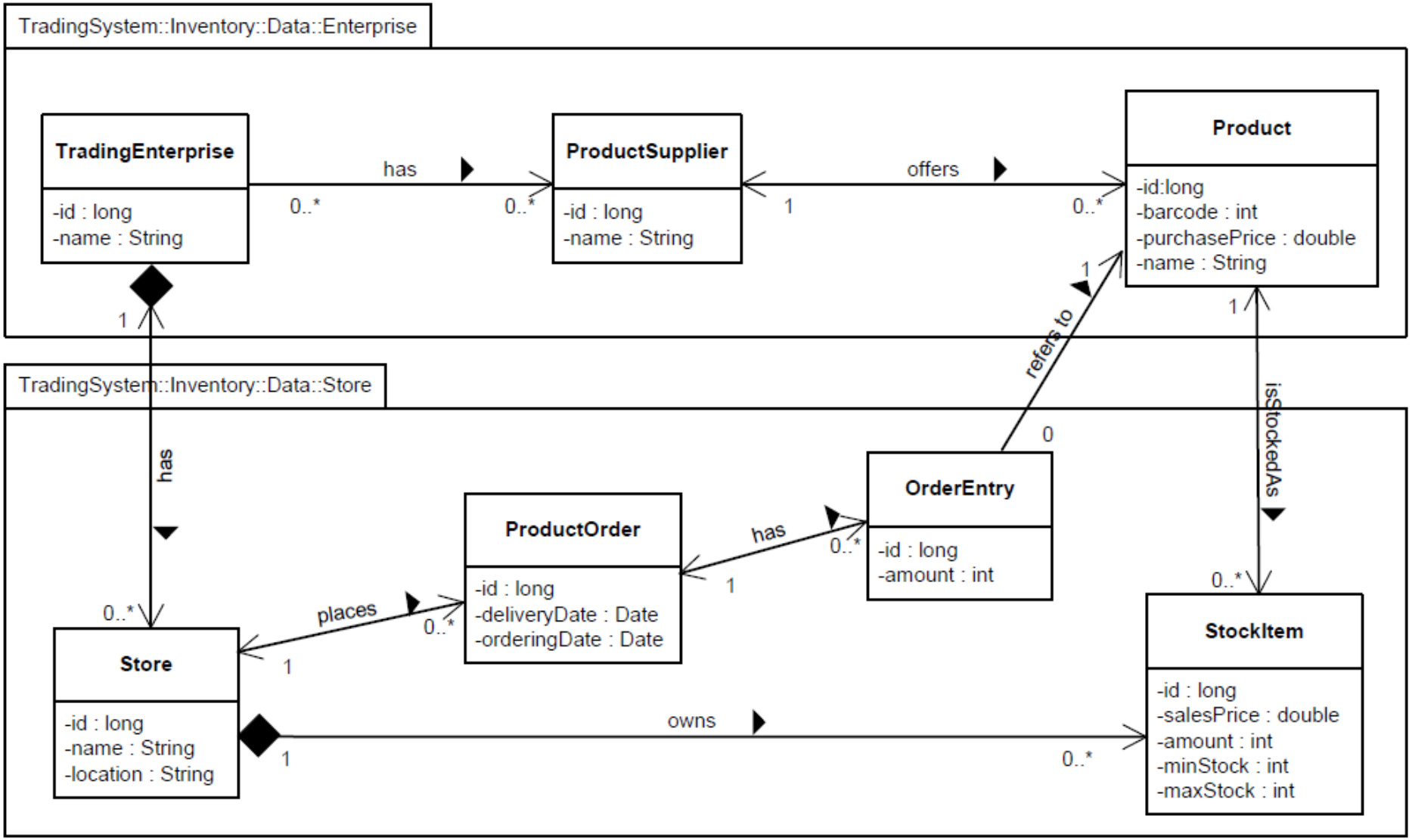
No data references are passed. Instead, so called *Transfer Objects (TO)* are defined and used for the data transfer.



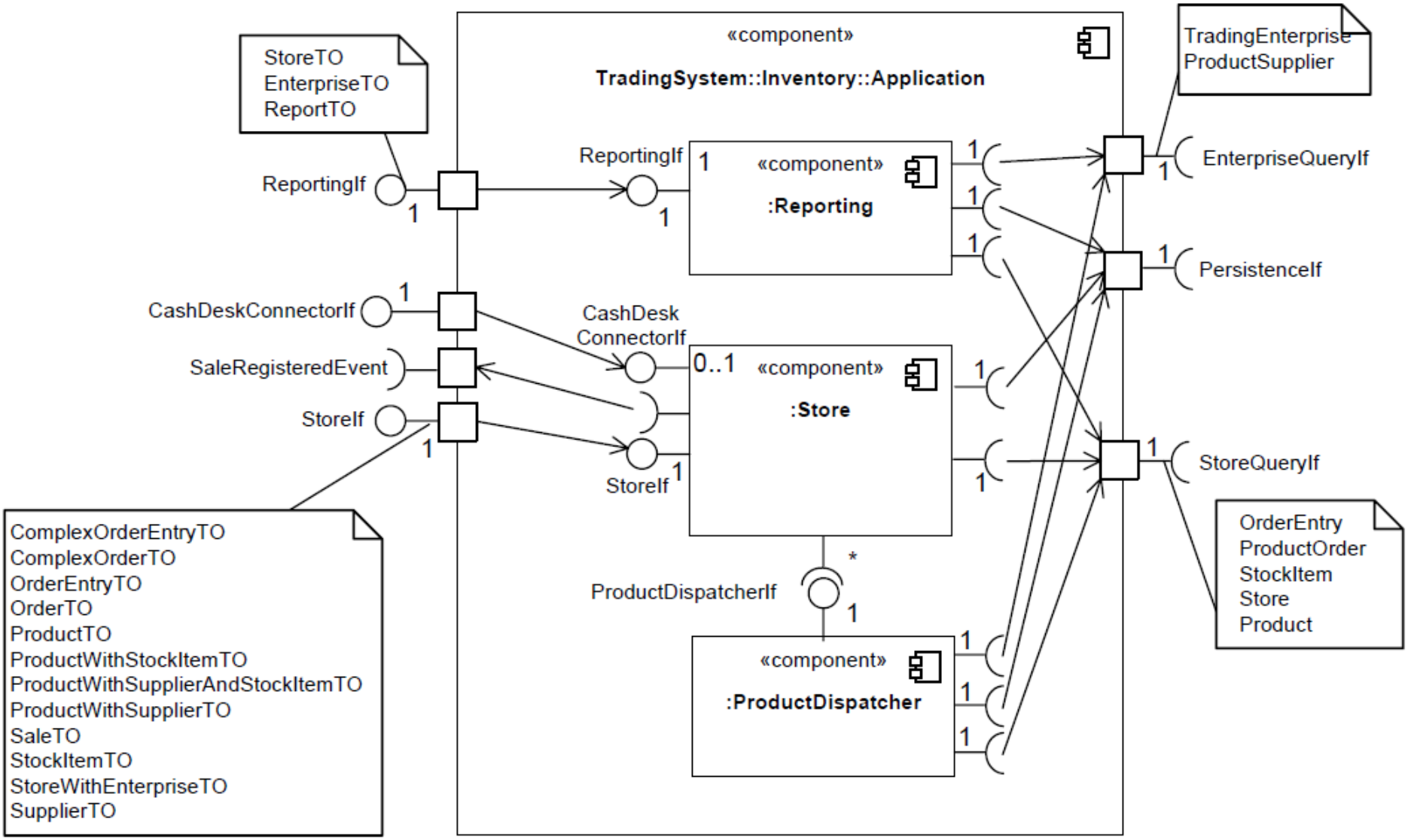
Inner structure of the Data layer of the component Inventory



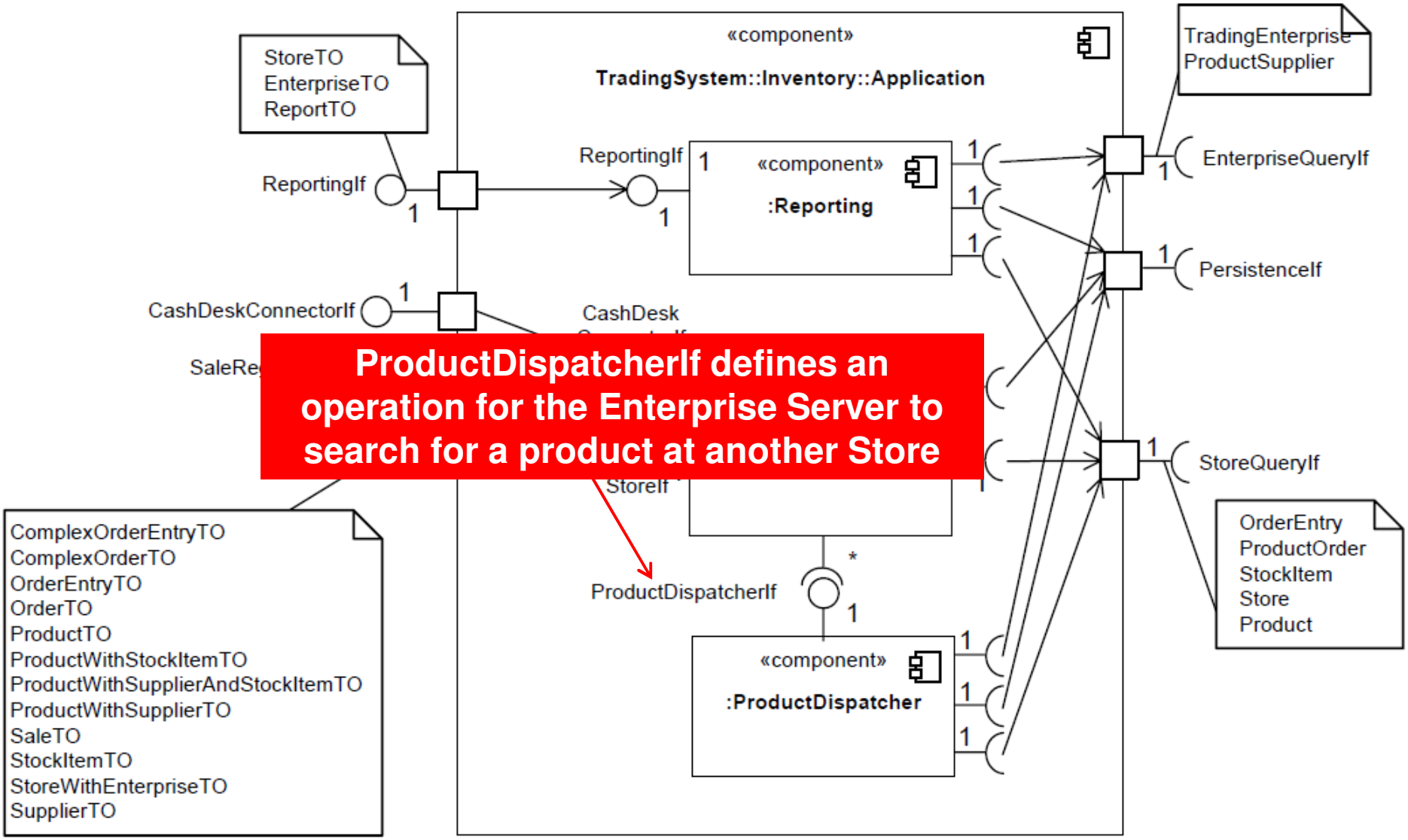
The data model of the Trading System



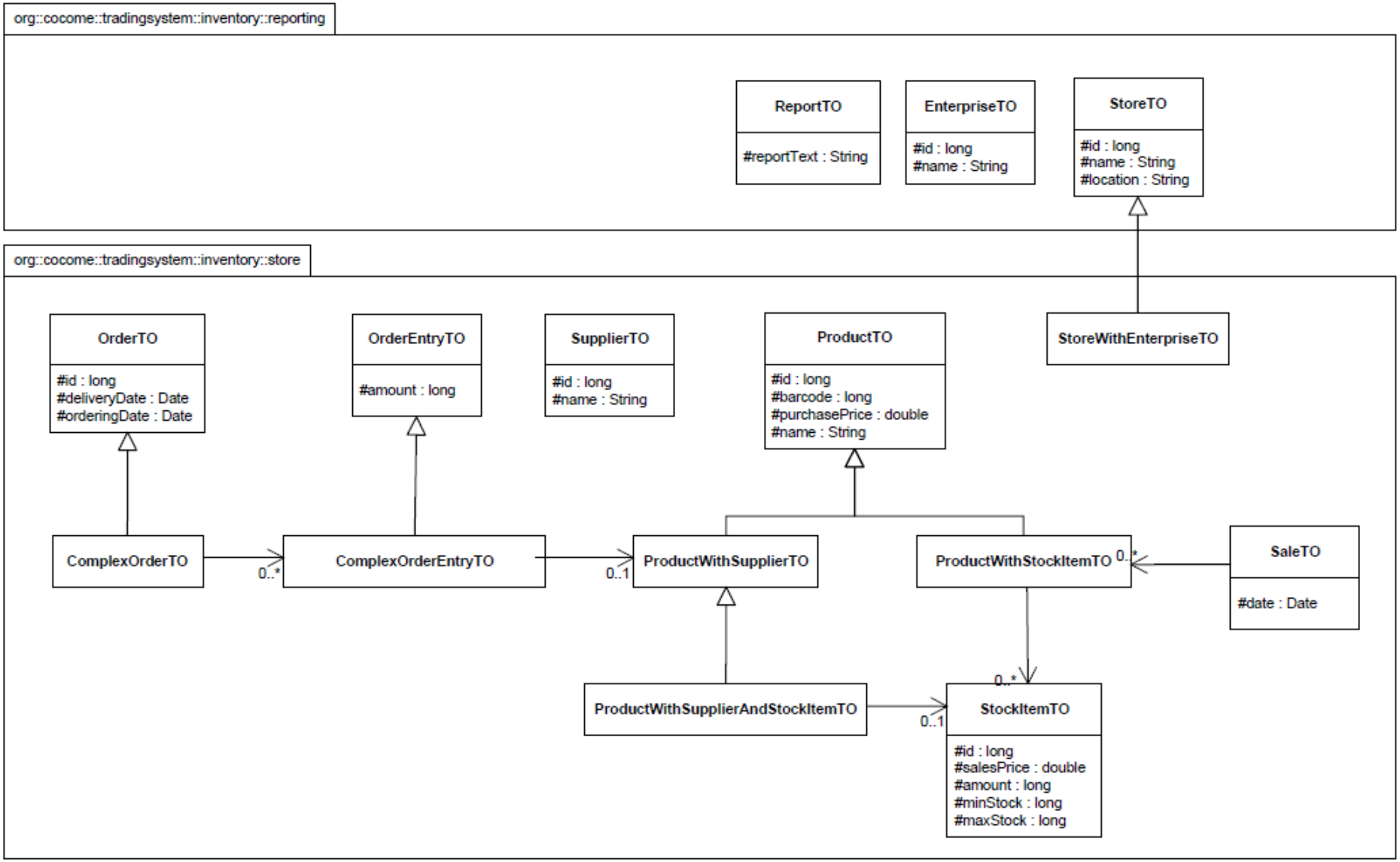
Inner structure of the Application layer



Inner structure of the Application layer

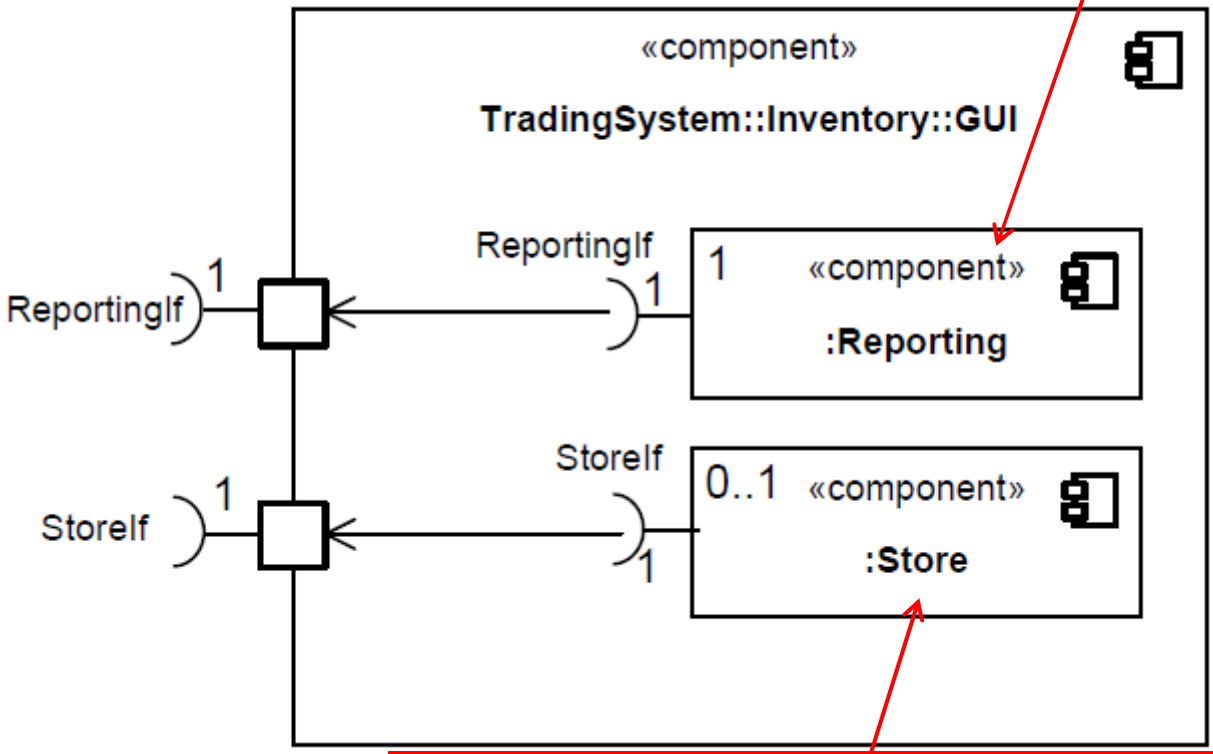


Overview of all Transfer Objects



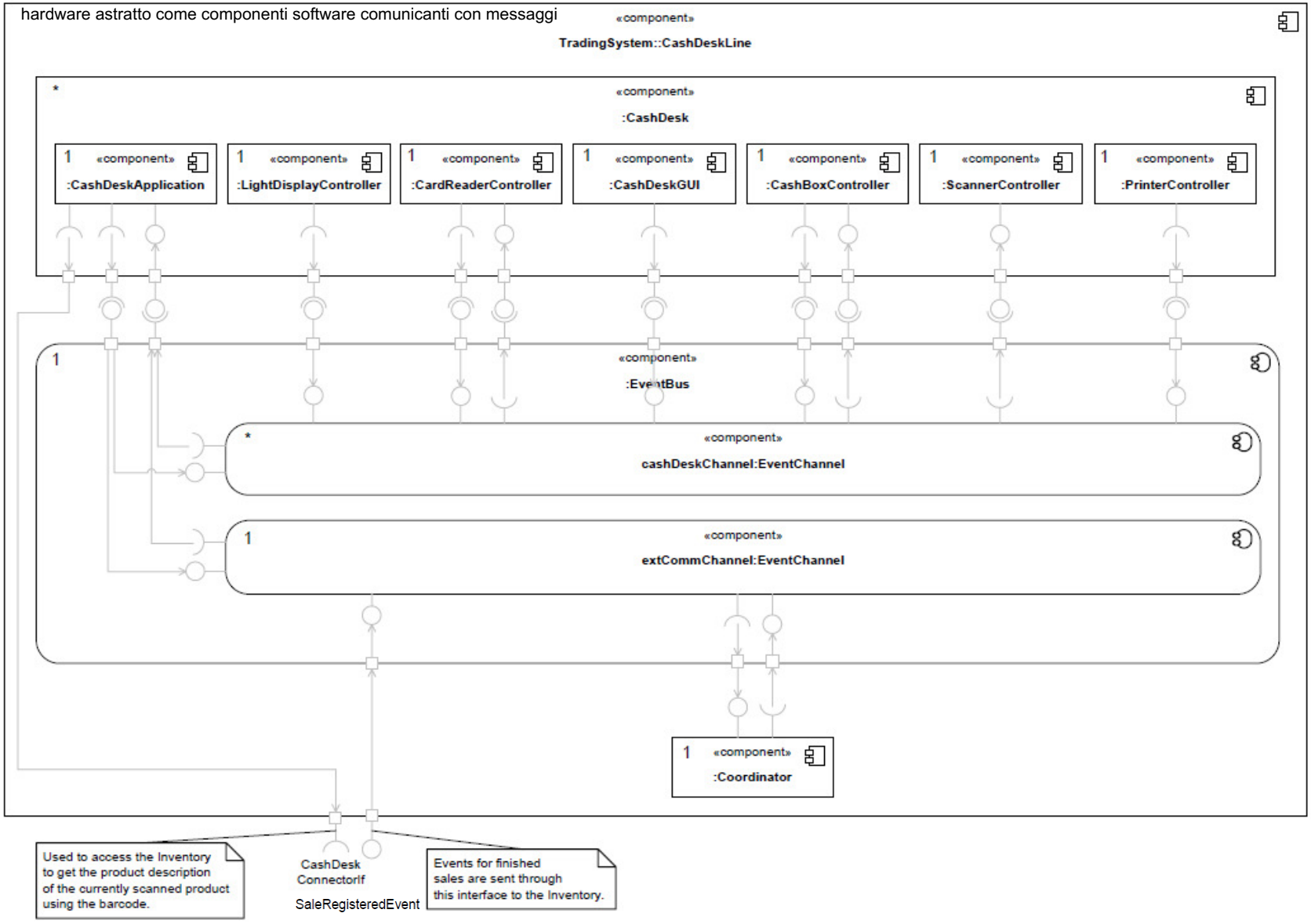
Inner structure of the GUI layer

For the visualization of various kinds of reports using the interface ReportingIf to get the data

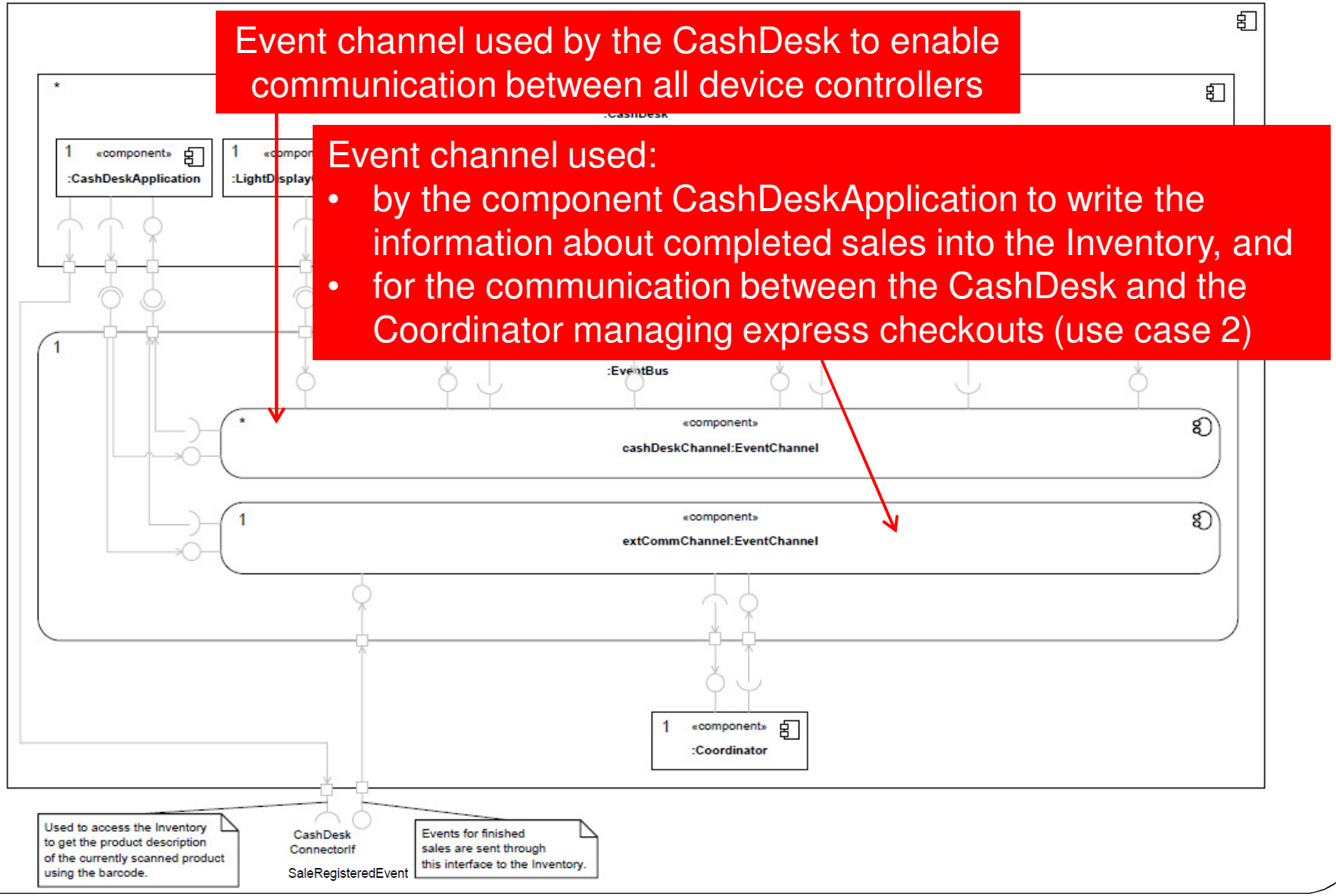


For the Store Manager in order to do managing tasks like ordering products or changing the sale prices

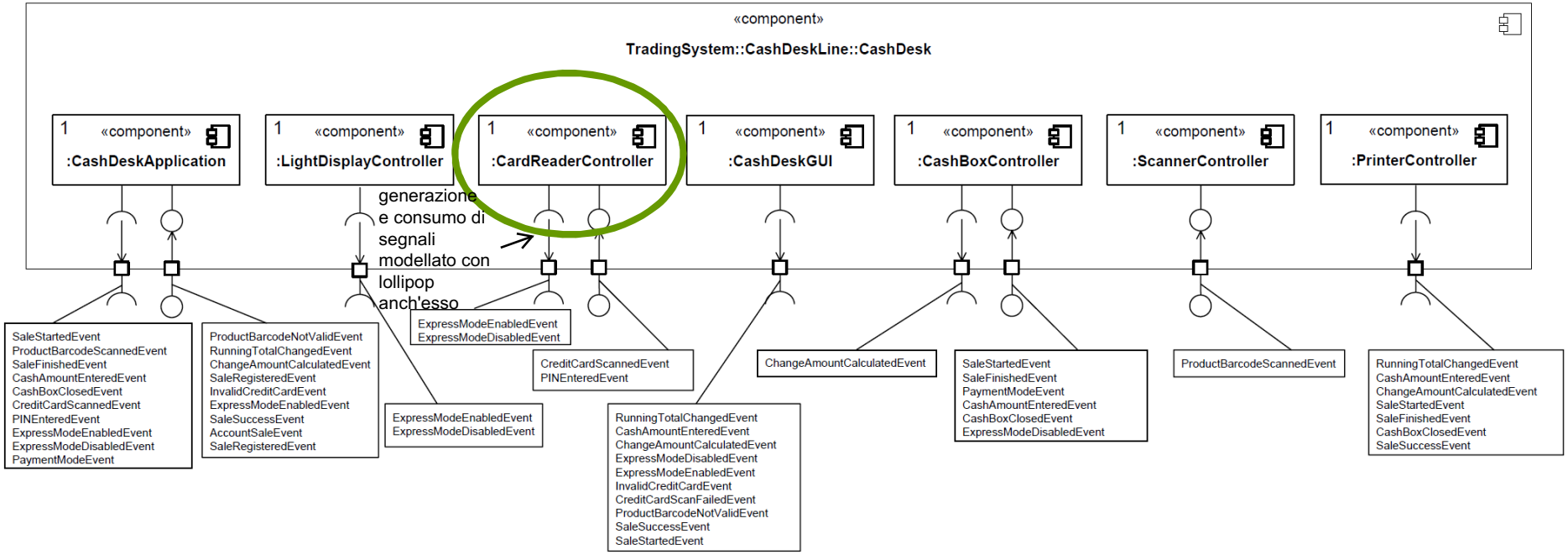
Structural view of the component CashDeskLine (the embedded part!)



Structural view of the component CashDeskLine (the embedded part!)



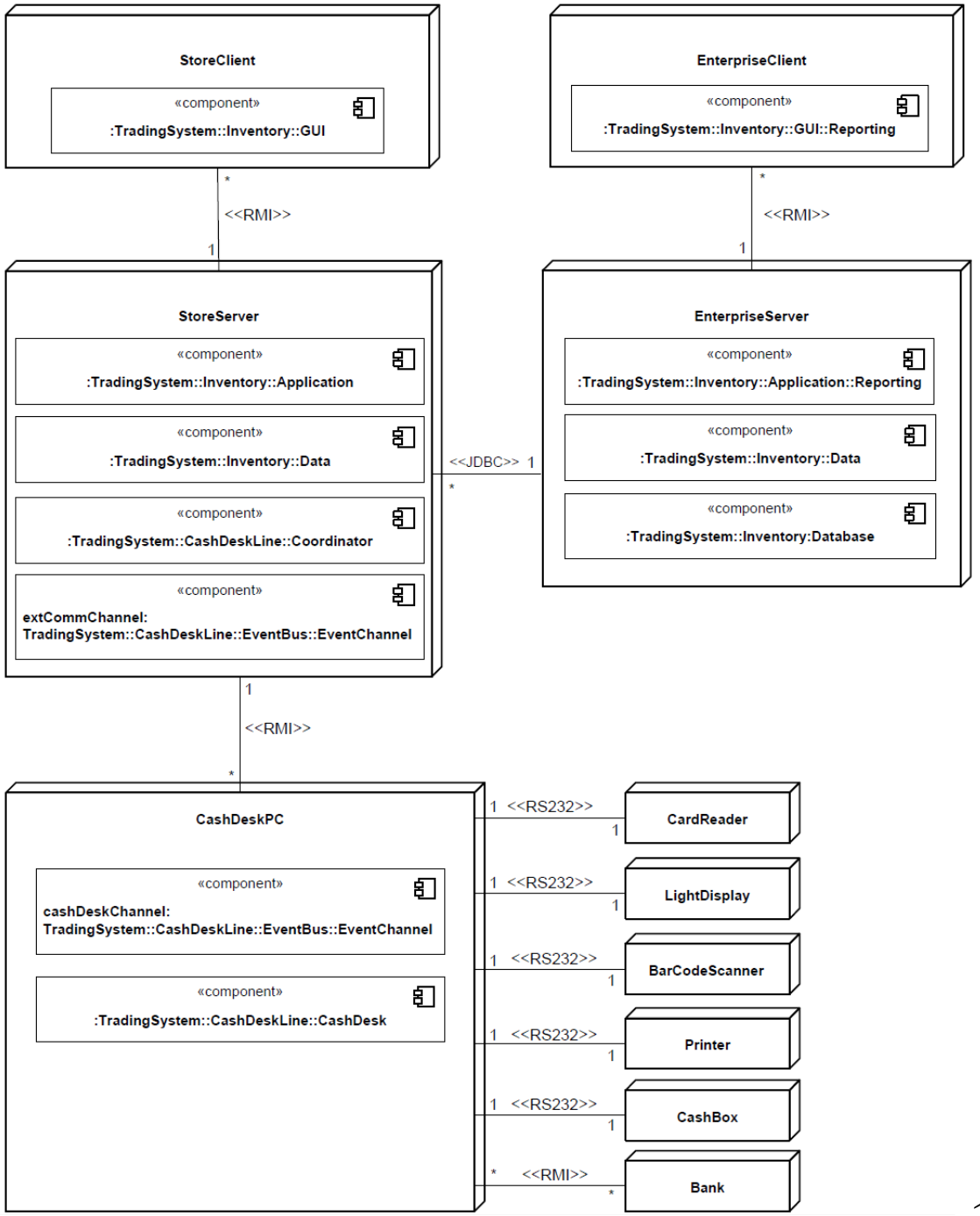
Detailed view on the component CashDesk



- ***Publish/subscribe messaging pattern***: events each component sends and for which types of events each component is registered at the channel

  modella la comunicazione asincrona che è tipica delle componenti hardware
- **Example**: the controller *CardReaderController*
  - sends the event *ExpressModeEnabledEvent* and *ExpressModeDisabledEvent*
  - handles the events *CreditCardScannedEvent* and *PINEnteredEvent*

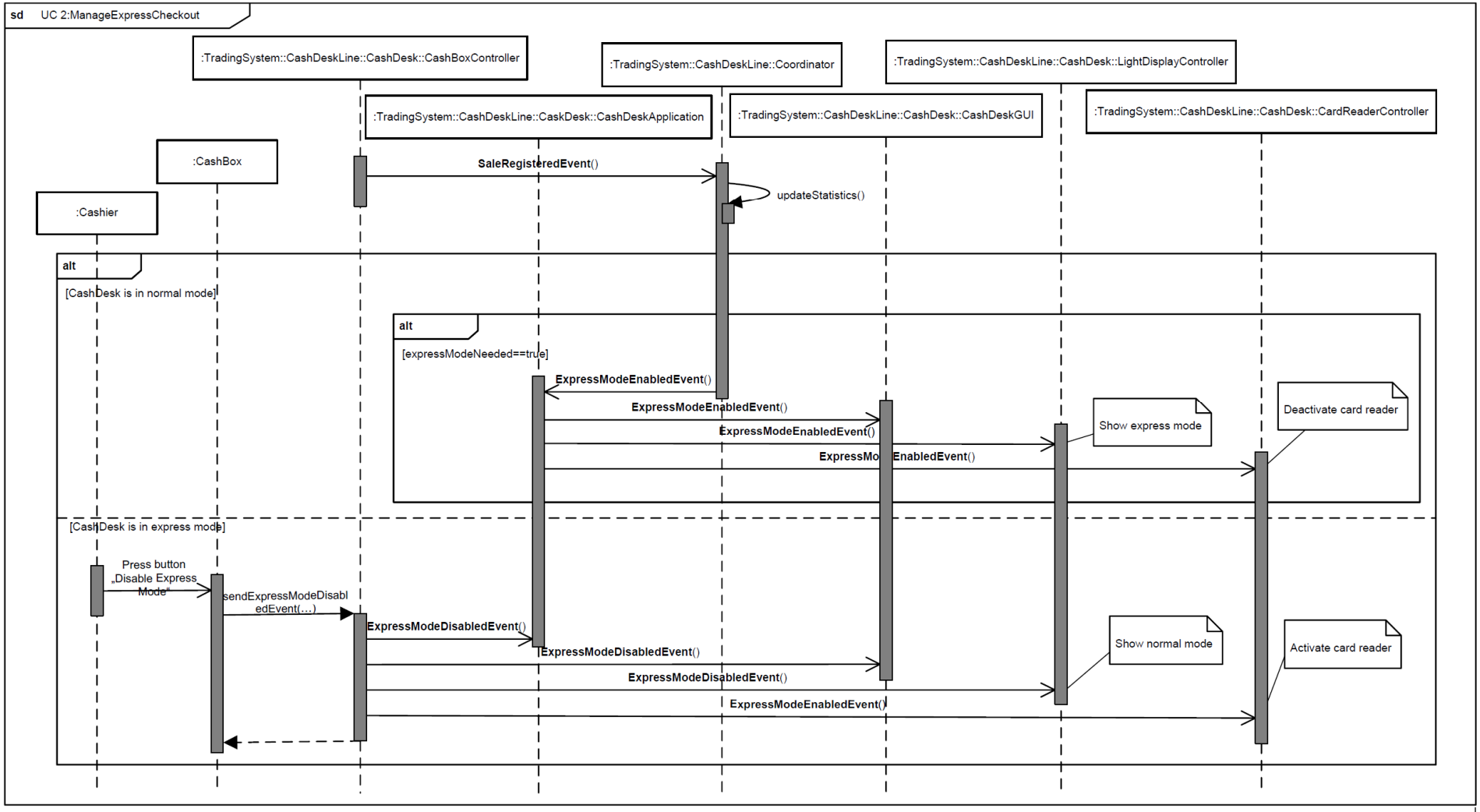
Deployment View of the Trading System



Scenario view of the Trading System

- Realization of each use case by using UML 2.0 sequence diagrams to show the interaction between actors and components
  - synchronous method calls are depicted using filled arrowheads
  - asynchronous method calls are depicted using unfilled arrowheads
- See for example, Behavioral View on UC 2 - Manage Express Checkout
  - Fig. 18 in the pdf file

Scenario View on UC 2 - Manage Express Checkout



CoCoME: Implementation aspects (Sect. 1.5)

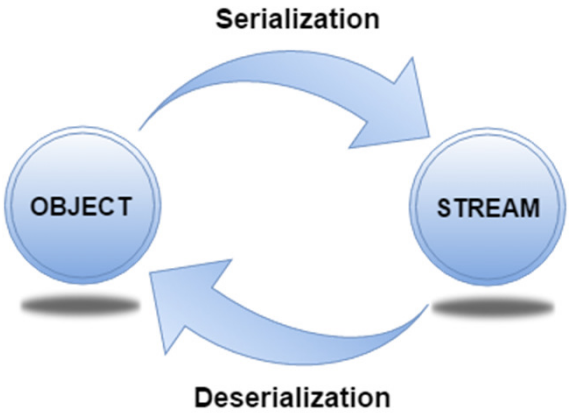
CoCoME – Java source code

https://www.cocome.org/downloads.html

- System interaction based on RMI and JMS technologies
- In addition to the Java source code:
  - Documentation in Javadoc
  - Test cases
  - Extra-functional requirements

Transfer Objects (1)

- *Data Transfer Object* (DTO or) is an object that carries data between distributed processes
  - It's a design pattern
- A DTO does not have any behavior except for storage and retrieval of its own data
- DTOs are often used in conjunction with data access objects to retrieve data from a database
- In Java a DTO is instance of a <u>class implementing the Java interface</u> **Serializable** to realize the *serialization*
  - writing the state of an object into a byte stream
- The reverse operation of serialization is called deserialization



Transfer Objects (2)

- Example

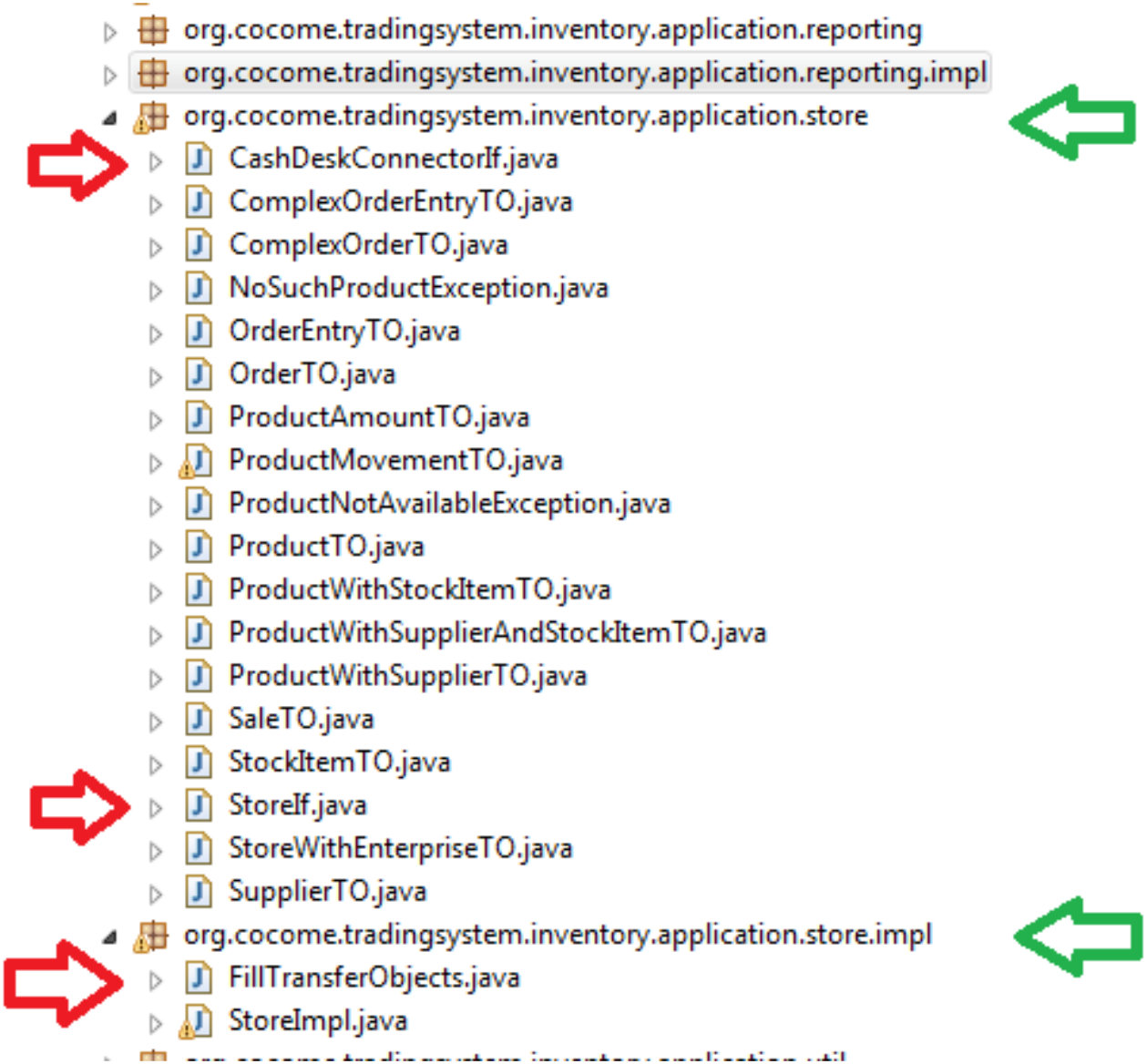
```
public class DataSerializzabile implements Serializable{
  private static final long serialVersionUID =
                                    5874806761123366899L;

  private int giorno;
  private int mese;
  private int anno;

  public void setGiorno(int g){ this.giorno = g; }
  public void setMese(int m){ this.mese = m; }
  public void setAnno(int a){ this.anno = a; }

  public int getGiorno(){ return this.giorno; }
  public int getMese(){ return this.mese; }
  public int getAnno(){ return this.anno; }
}
```

Store Component (2)



Store Component (3)

- Provided interface **CashDeskConnectorIf**

org.cocome.tradingsystem.inventory.application.store.CashDeskConnectorIf

```
public interface CashDeskConnectorIf extends Remote {

/** . . . */
void bookSale(SaleTO saleTO) throws RemoteException;

/** . . . */
ProductWithStockItemTO getProductWithStockItem
                                        (long productBarCode)
              throws NoSuchProductException, RemoteException;
}
```

Store Component (4)

- Provided interface **StoreIf**
- `org.cocome.tradingsystem.inventory.application.store.StoreIf`

```
public interface StoreIf extends Remote {

StoreWithEnterpriseTO getStore() throws RemoteException;

List<ProductWithStockItemTO> getProductsWithLowStock()
                                        throws RemoteException;

/*. . . */
List<ProductWithSupplierTO> getAllProducts() throws
RemoteException;

ComplexOrderEntryTO[] getStockItems(
                            ProductTO[] requiredProductTOs)
                                        throws RemoteException;

}
```

Implementation of the component Store

```
public class StoreImpl extends UnicastRemoteObject implements
StoreIf, CashDeskConnectorIf {

  //From interface StoreIf
  public ProductWithStockItemTO changePrice(StockItemTO
       stockItemTO){ /*. . . */ }

  //From interface StoreIf
  public List<ProductWithSupplierTO> getAllProducts() { /* */}

  /* . . . */

  //From interface CashDeskConnectorIf
  public void bookSale(SaleTO saleTO) { /* . . . */ }

  //Private
  private void checkForLowRunningGoods(){ /* . . . */ }

}
```

Required interfaces of the component Store

Required interfaces

```
public class StoreImpl extends UnicastRemoteObject implements
                                        StoreIf, CashDeskConnectorIf {

  private StoreQueryIf storequery =
              DataIfFactory.getInstance().getStoreQueryIf();

  private EnterpriseQueryIf enterpriseQuery =
       DataIfFactory.getInstance().getEnterpriseQueryIf();

  private PersistenceIf persistmanager =
       DataIfFactory.getInstance().getPersistenceManager();

  //. . .
}
```

The *Factory pattern* is used to create references to the required components

Compilation

- Compilation through *ant*
  - Unzip the sorce code cocome-impl in a directory (the pathname must not contain spaces), e.g. `c:\cocome-impl`
  - In `c:\cocome-impl\rsc` execute the command `ant compile` to compile the overall application
- Javadoc generation
  - In `c:\cocome-impl\rsc` execute the command `ant doc`
  - Output directory: `c:\cocome-impl\doc`

Installation (1)

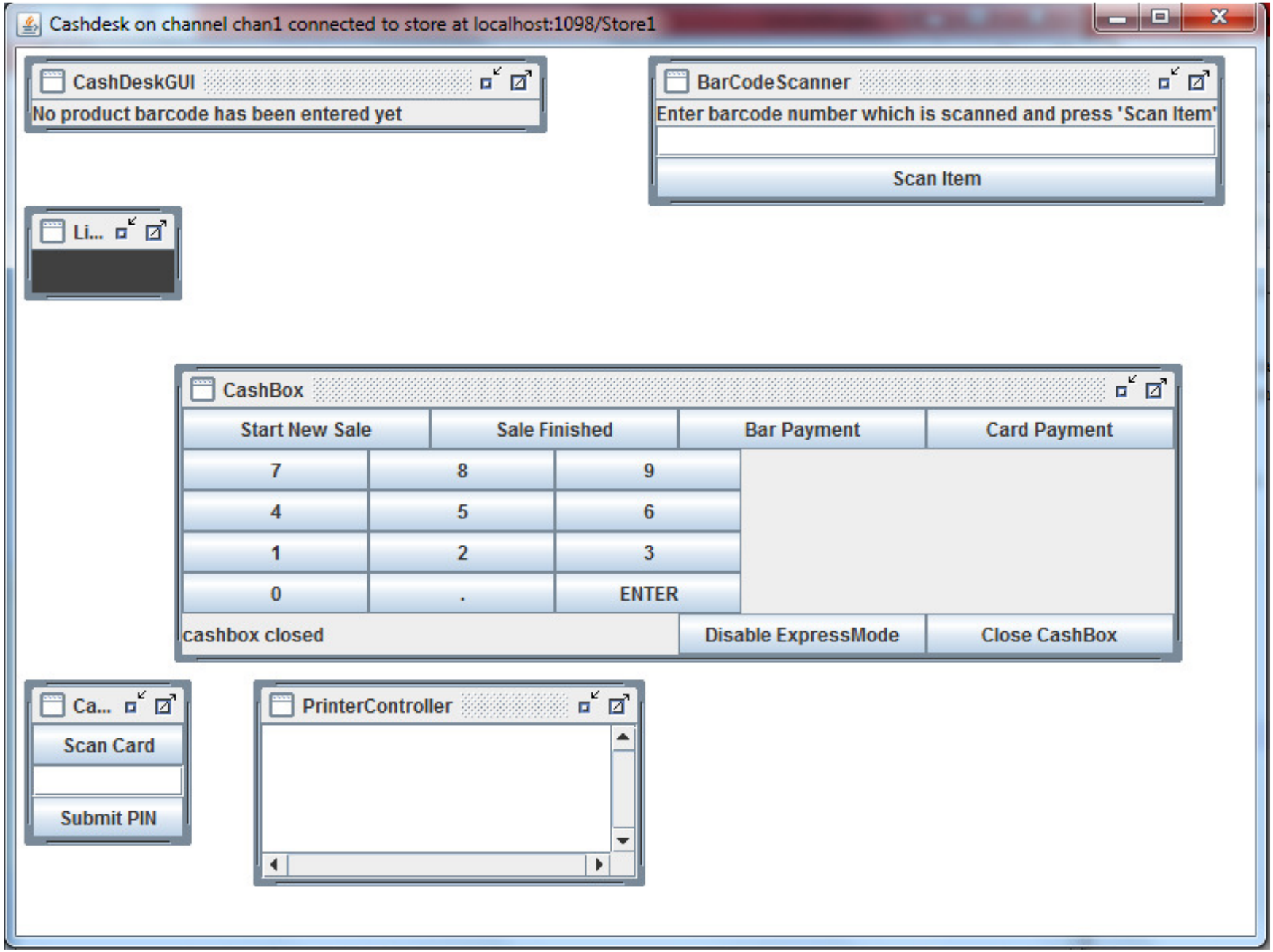
- Infrastrutture:
  - `ant runInfrastructure/ant stopInfrastructure`: start/stop the message broker, RMI and the database
  - `ant fillDB` : to initialize the database with some test data
    - **ATTENTION!:** runInfrastructure must be executed first!
  - `ant deleteDB`: to eliminate the database
    - **ATTENTION!**: stopInfrastructure must be executed first!

Installation (2)

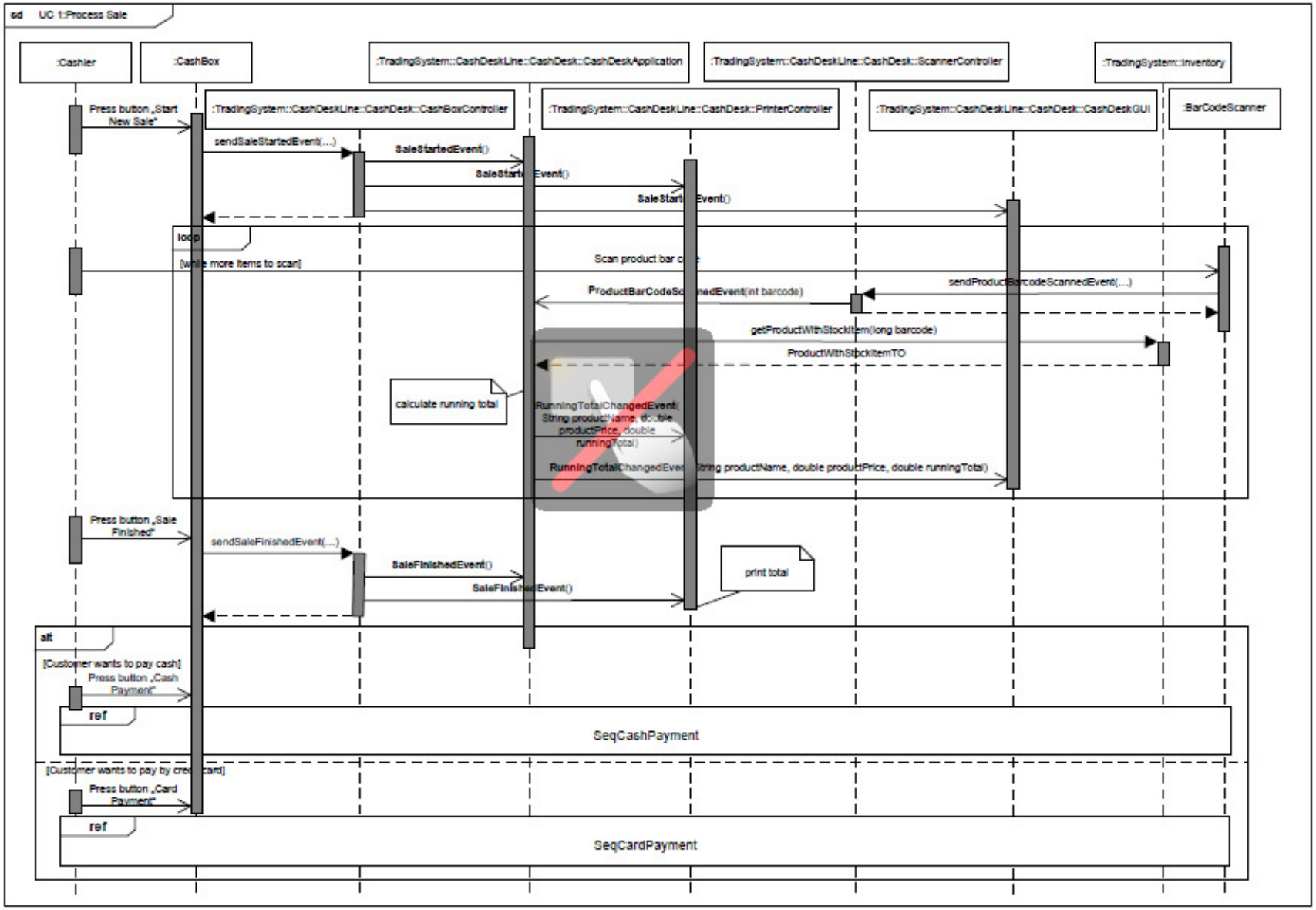
- Inventory
  - `ant runInventorySystem`
  - `ant runInventorySystemGUIs`
- Cash Desk
  - `ant runCashDeskLines`

Run the application

1. ant deleteDB
2. ant runInfrastructure
3. ant fillDB
4. ant runInventorySystem
5. ant runCashDeskLines


Cashdesk on channel chan1 connected to store at localhost:1098/Store1
CashDeskGUI
No product barcode has been entered yet
BarCodeScanner
Enter barcode number which is scanned and press 'Scan Item'
Scan Item
Li...
CashBox
Start New Sale
Sale Finished
Bar Payment
Card Payment
7
8
9
4
5
6
1
2
3
0
.
ENTER
cashbox closed
Disable ExpressMode
Close CashBox
Ca...
Scan Card
Submit PIN
PrinterController

Sequence diagram – UC1-Process sale



Unit Testing

- Test cases are based on use cases
  - Junit tests from use case steps/sequence diagrams
  - See project `cocome-systests`

Test cases examples

- **ProcessSaleCash**
  - **USE CASE**: UC1 – Process Sale
  - **DESCRIZIONE**: Buy products paying by cash. The test ends with success if no exceptions are arisen during executions.

- **ProcessSaleCreditCard**
  - **USE CASE**: UC1 – Process Sale
  - **DESCRIZIONE**: Buy products paying by credit card. The test ends with success if no exceptions are arisen during executions.