

Project Deliverable D5.2

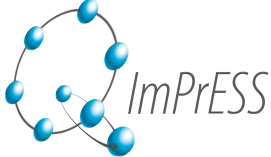
Experiments with impact analysis

Project name: Q-ImPrESS
Contract number: FP7-215013
Project deliverable: D5.2: Experiments with impact analysis
Author(s): Jan Kofroň, Pavel Parízek
Work package: WP5
Work package leader: MDU
Planned delivery date: M28
Delivery date: M28
Last change: 29.04.2010
Version number: 1.0

Abstract

This document describes experimental evaluation of the method for checking consistency between service implementation in Java and its behaviour model in TBP. All experiments are performed on the CoCoME application.

Keywords: consistency checking, case study

	D5.2: Experiments with impact analysis	
	Version: 1.0	Last change: 29.04.2010

Revision history

Version	Change date	Author(s)	Description
0.1	10.03.2010	Kofron	Doc structure
0.2	30.03.2010	Parizek	Initial content
0.3	07.04.2010	Parizek	Added abstract, conclusion and small changes
0.4	23.04.2010	Kofron	Fixes, completing text, adding appendices
0.5	28.04.2010	M. Trifu	Review
1.0	29.04.2010	Kofron	Final updates according to review comments

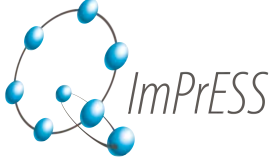
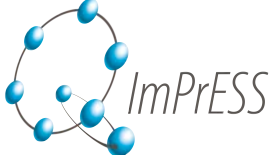
	D5.2: Experiments with impact analysis	
	Version: 1.0	Last change: 29.04.2010

Table of contents

1	Introduction	4
2	Background	4
2.1	<i>CoCoME</i>	4
2.2	<i>TBP</i>	6
3	Evolution scenarios.....	7
3.1	<i>Adding new method.....</i>	7
3.2	<i>Adding a required interface and a method call</i>	8
3.3	<i>Adding new component and updating bindings</i>	10
4	Impact of changes on the code.....	10
5	Conclusion	12
6	Glossary	13
7	References	14
8	Appendix	15
8.1	<i>Modified implementation of the CashDeskApplicationImpl.....</i>	15
8.2	<i>Original implementation of the CashDeskApplicationImpl</i>	22
8.3	<i>Source code of SecurityManagerIf interface</i>	29
8.4	<i>Modified implementation of CashDeskAppEventHandlerIf</i>	30
8.5	<i>Original implementation of CashDeskAppEventHandlerIf.....</i>	31

	D5.2: Experiments with impact analysis	
	Version: 1.0	Last change: 29.04.2010

1 Introduction

This document describes our experience with the methods and tools for checking consistency between the implementation of a service and its behaviour model, which are described in D5.1. We performed experiments that capture common evolutionary changes, like adding a new component into the system or removing a method call, and their impact onto the corresponding code. Experimenting and tuning the Q-ImPRESS method using the demonstrators might, however, result in a situation that the method works well for the demonstrators, but not for other software. Therefore, in order to preserve the significance of application of the Q-ImPRESS method on the demonstrators (validation), we have chosen another application for the experiments. The evolutionary changes that we have captured are described in Chapter 3, and the results of our experiments are presented and discussed in Chapter 4.

2 Background

This chapter describes the Common Component Modelling Example (CoCoME) [1], which has been used to perform the experiments with evolutionary changes and impact analysis, as well as Threaded Behaviour Protocols (TBP) [2] – a behaviour specification formalism used to model the behaviour of components of the system under consideration.

2.1 CoCoME

The CoCoME application [1] is a prototype of a trading system for supermarkets. It is a software system built from hierarchical components. The architecture of CoCoME is shown in Figure 1. It consists of two parts: (i) an inventory subsystem and (ii) a cash desk line.

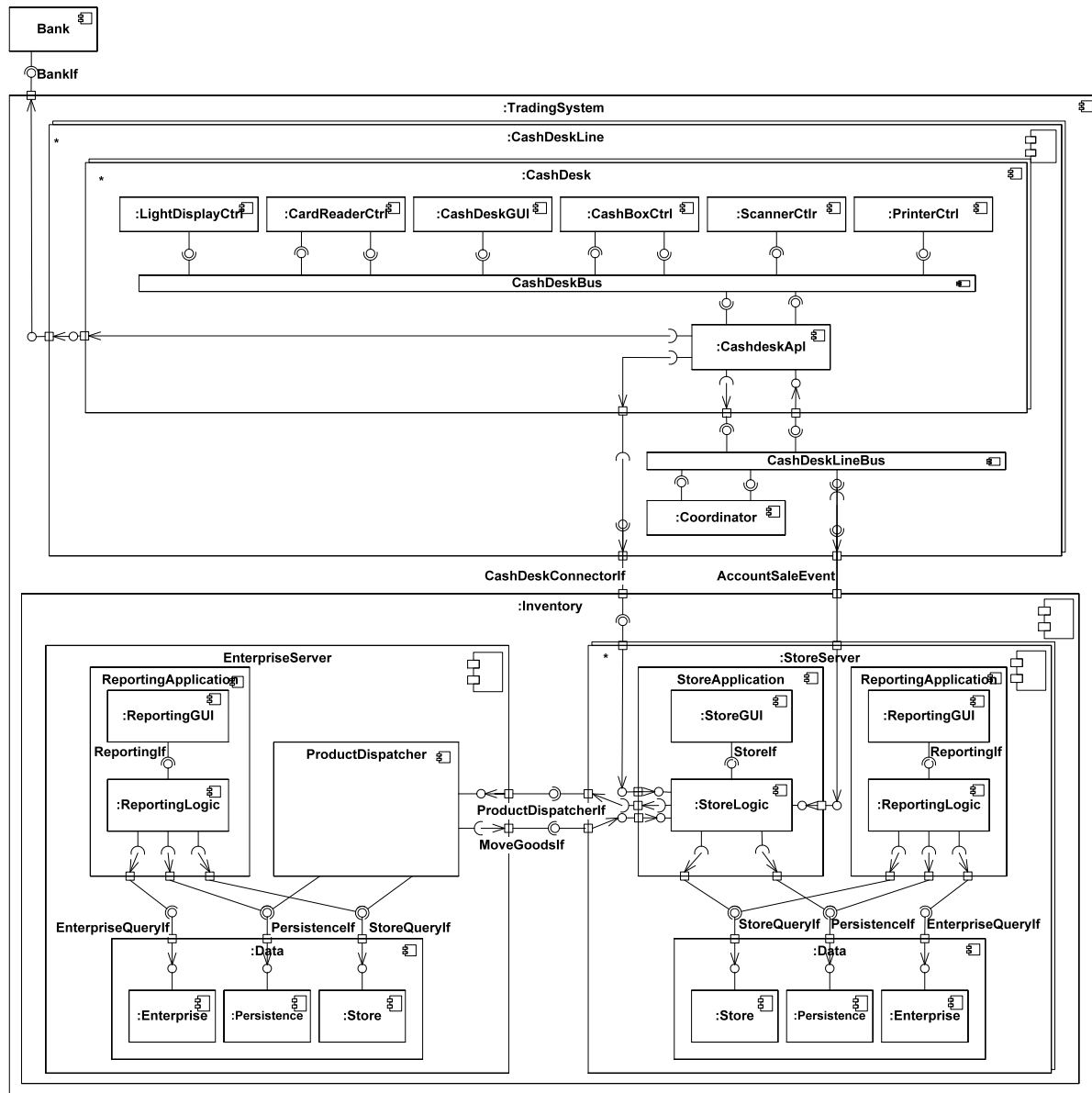
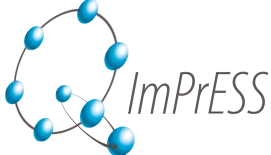


Figure 1: Architecture of CoCoME

The inventory subsystem is responsible for management of products, items, and orders. It consists of two layers of databases – one belonging to a single store (the *StoreServer* component) and another at the level of the whole enterprise (the *EnterpriseServer* component). This subsystem also provides reporting functionality for the managers.

A cash desk line is formed by a set of cash desks. Each cash desk in a line is represented by several components that control cash desk hardware (e.g., a bar code scanner and a credit card reader). Although the number of cash desks in a system can be arbitrary, for the purpose of experiments we used a configuration with a single cash desk.

In this document, we perform all experiments on the *CashDeskApplication* component and its immediate context (components directly connected to it). The experiments focus on analysis how a particular change corresponding to an evolution scenario described in D1.1 affects the source code of the corresponding component and whether the possible

	D5.2: Experiments with impact analysis	
	Version: 1.0	Last change: 29.04.2010

inconsistency between the behaviour model and component implementation is detected by the tools. `CashDeskApplication` is responsible for control of a cash desk – it handles the following tasks related to sales: payments by customers (cash or credit card), switch to the express checkout mode (when the customer buys only a few goods and pays cash), and printing of bills. This component represents the software part of a cash desk. All the components of the CoCoME application are implemented in Java.

2.2 TBP

The formalism of Threaded Behaviour Protocols (TBP) [2] allows specifying the externally observable behaviour of a service – valid sequences of method calls on the interfaces of a service. For illustration, fragment of the TBP specification of the `CashDeskApplication` component is listed in Figure 2.

```

component CashDeskApplication {
  types {
    states = {
      INITIALIZED, SALE_STARTED, SALE_FINISHED, ...
    }
  }

  vars { states state = INITIALIZED }

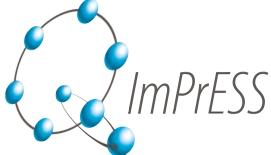
  provisions {
    main {
      (
        ?CashDeskApplicationEventHandlerIf.onSaleStartedEvent()
        +
        ?CashDeskApplicationEventHandlerIf.onSaleFinishedEvent()
        +
        ...
      ) *
    }
  }

  reactions {
    CashDeskApplicationEventHandlerIf.onSaleStartedEvent()
    {
      switch (state) {
        INITIALIZED: { state <- SALE_STARTED }
        default: { NULL }
      }
    }
    ...
  }
}

```

Figure 2: TBP specification of `CashDeskApplication`

The protocol describes (i) the expected sequences of input messages from the keyboard and barcode scanner (the `provisions` section) and (ii) responses of the component to the input

	D5.2: Experiments with impact analysis	
	Version: 1.0	Last change: 29.04.2010

messages received via calls on required interfaces (in the `reactions` section). Responses include printing messages on a display, handling the credit card reader, querying the inventory subsystem for information about goods, and updating the numbers of available items. A more complete description of TBP can be found in Section 2.4 of the D5.1 document.

3 Evolution scenarios

In this chapter, we describe the evolutionary changes of the behaviour model (in TBP) of a single service, which we consider. They reflect the scenarios for system evolution that were identified by the industrial partners as required – the scenarios are described in the requirements document D1.1, section 4.4.3.

We consider the following specific evolutionary changes (corresponding categories defined in the D1.1 Requirements document are listed in parentheses):

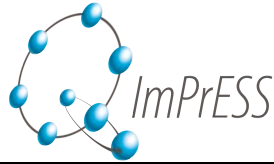
- Add or remove a method to/from the component (service evolution);
- Add or remove a required interface to/from the component (service evolution);
- Add or remove a method call on an existing required interface to/from an existing method of the component (service evolution);
- Add a new component that implements a required interface of another component and modify bindings accordingly (new service, service wiring).

We illustrate the meaning of these evolutionary changes by several examples involving the `CashDeskApplication` component. We show only the addition changes, since it is obvious that the corresponding remove changes will be performed in a similar fashion.

3.1 Adding new method

Let us suppose that we want to update the cash desk so that it cannot be used without being properly initialized. This should be performed at a single point, when a cashier comes to his or her cash desk and starts to use it by inserting his or her key into the lock. This action fires an event resulting in call of the `onCashDeskInitEvent` method on the `CashDeskApplicationEventHandlerIf` interface.

We show the way to add the `onCashDeskInitEvent` method into the component's TBP behaviour model. The method initializes the cash desk so that it can process sales by changing its "mode" from `NONE` to `INITIALIZED`. The behaviour model of the component is updated as illustrated on Figure 3 (modified fragments are shown in bold), while the original and modified implementations of the `CashDeskApplication` component can be found in Appendix 8.2 and 8.1, respectively.



```
component CashDeskApplication {
  types {
    states = {
      NONE, INITIALIZED, SALE_STARTED, SALE_FINISHED, ...
    }
  }

  vars { states state = NONE }

  provisions {
    main {
      (
        ?CashDeskApplicationEventHandlerIf.onCashDeskInitEvent()
        +
        ?CashDeskApplicationEventHandlerIf.onSaleStartedEvent()
        +
        ?CashDeskApplicationEventHandlerIf.onSaleFinishedEvent()
        +
        ...
      ) *
    }
  }

  reactions {
    CashDeskApplicationEventHandlerIf.onCashDeskInitEvent()
    {
      switch (state) {
        NONE: { state <- INITIALIZED }
        default: { NULL }
      }
    }

    CashDeskApplicationEventHandlerIf.onSaleStartedEvent()
    { ... }

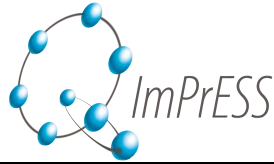
    ...
  }
}
```

Figure 3: Adding new method into TBP behaviour model

The change involved adding a new value to the `states` type, acceptance event of the method into the `provisions` section, and the method body into the `reactions` section. Note that if the `onCashDeskInitEvent()` is invoked later, i.e., in a state different from `NONE`, it is ignored via the `{ NULL }` reaction in the default branch of the switch statement.

3.2 Adding a required interface and a method call

Yet another improvement as to the security of the cash desk might be that, in addition to insertion of the cashier's key, he or she has to provide a PIN. Since there are many cash desks inside each store and a cashier can work at any of these, it is natural to have a central



component that would store and validate the entered information. To implement this change, we extend the `CashDeskApplication` component by addition of a new required interface. Since addition of an empty required interface makes not much sense, we also add a new required method of the interface.

We show the way to add a call of the `checkCredentials` method on a new required interface `SecurityManagerInterface` (Appendix 8.3) into the TBP behaviour model. The method accepts the cash desk assistant's PIN, and checks its validity – this operation is performed during initialization of the cash desk. The updated behaviour model of the component is in Figure 4 (modified fragments are in bold), while the updated source code can be found in Appendix 8.1.

```
component CashDeskApplication {
  types {
    states = {
      NONE, INITIALIZED, SALE_STARTED, SALE_FINISHED, ...
    }
  }

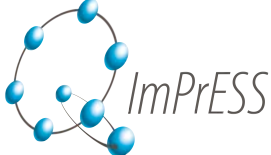
  vars { states state = NONE }

  provisions {
    main {
      (
        ?CashDeskApplicationEventHandlerIf.onCashDeskInitEvent()
        +
        ?CashDeskApplicationEventHandlerIf.onSaleStartedEvent()
        +
        ?CashDeskApplicationEventHandlerIf.onSaleFinishedEvent()
        +
        ...
      ) *
    }
  }

  reactions {
    CashDeskApplicationEventHandlerIf.onCashDeskInitEvent()
    {
      switch (state) {
        NONE: {
          !SecurityManagerInterface.checkCredentials();
          ( (state <- INITIALIZED) + NULL )
        }
        default: { NULL }
      }
    }
    ...
  }
}
```

Figure 4: Adding required interface and method call into the TBP model

The change involved adding call of the `checkCredentials` method into the reaction for the `onCashDeskInitEvent` method.

	D5.2: Experiments with impact analysis	
	Version: 1.0	Last change: 29.04.2010

3.3 Adding new component and updating bindings

The component providing validation of the cashier's credentials needs to be also added to the service architecture. The newly added component is named `SecurityManager`. After a new component is created, i.e., the static structure has been changed, its behaviour model in TBP has to be also defined and the component itself has to be implemented. The changes employed by this change in the code and behaviour model of existing components are described in the previous sections; we now have to define a TBP model for a new component `SecurityManager`, which implements the `SecurityManagerInterface` interface (Appendix 8.3). The behaviour model of the `SecurityManager` component is listed in Figure 5.

```

component SecurityManager {
  types {}
  vars {}

  provisions
  {
    main {
      (?SecurityManagerInterface.checkCredentials()) *
    }
  }

  reactions
  {
    SecurityManagerInterface.checkCredentials() { NULL }
  }

  threads {}
}

```

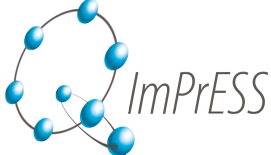
Figure 5: Behaviour model of the `SecurityManager` component in TBP

The newly introduced `SecurityManager` performs verification of credentials provided by a user, not requiring other components to perform this task. Therefore there is an empty reaction to invoking the `checkCredentials` method.

4 Impact of changes on the code

This chapter describes the results of our experiments with checking inconsistencies between service behaviour models in TBP and implementations in Java. We performed changes of the behaviour model of the `CashDeskApplication` component and tried to determine the impact of the changes on the Java source code using the consistency checking framework described in D5.1. Briefly, the framework consists of a modified Java PathFinder [3] and a tool able to process the behaviour model written in TBP. The consistency checking then involves exhaustive traversal of the state space of the Java code together with the state space of the behaviour model and checking for inconsistencies in corresponding states.

First, we modified the behaviour model of the component in TBP as described in Section 3.1 and applied the consistency checking framework on the modified TBP model and the original Java implementation. Results show that the checking framework correctly determined that the

	D5.2: Experiments with impact analysis	
	Version: 1.0	Last change: 29.04.2010

onCashDeskInitEvent method is missing in the source code (see a fragment of the error report in Figure 6).

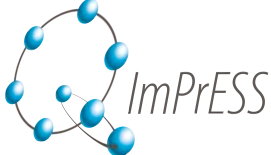
```
[java] JavaPathfinder v4.1 - (C) 1999-2007 NASA Ames Research Center
[java]
[java] ===== system under test
[java] application: cocome\CashDeskApp_main.java
[java]
[java] ===== search started: 1.4.10 15:46
[java]
[java] ===== error #1
[java] org.ow2.dsrg.fm.tbpjava.checker.PropertyUncaughtExceptions
      ("Tested method throw exception. Typically it is bad method
       call (illegal parameter or invalid call time). Check
       provision definition (in TPB protocol) or parameter
       definition in cocome.TestValuesCashDeskApp class")
[java] java.lang.RuntimeException: Call on provided interface can't
      be generated
[java]     at cocome.CashDeskApp_main$1.run(CashDeskApp_main.java:45)
```

Figure 6: Fragment of error report provided by checking framework for new method

After fixing the Java source code of the component and modifying the behaviour model as described in Section 3.2, i.e., adding a required interface and a corresponding method call, we ran the checking framework again. This time the framework correctly determined that the SecurityManagerInterface interface and a call of the checkCredentials method are not present in the component's Java implementation. A fragment of the error report is listed in Figure 7.

```
[java] JavaPathfinder v4.1 - (C) 1999-2007 NASA Ames Research Center
[java]
[java] ===== system under test
[java] application: cocome\CashDeskApp_main.java
[java]
[java] ===== search started: 1.4.10 15:46
[java] CashDesk 1: Application: CashDeskInitEvent received
[java]
[java] ===== error #1
[java] org.ow2.dsrg.fm.tbpjava.checker.ProtocolListener
[java] Tested component is not compliant with the protocol
[java] ...
[java] Interface calls event trace
[java] Event type | Thread | IF type | Interface name | Method name
[java] -----
[java] Started threads :1(ENV_THREAD), 2(ENV_THREAD), 3(ENV_THREAD)
[java] call          | 1      | provided |
      CashDeskApplicationEventHandlerIf | onCashDeskInitEvent
[java] return        | 1      | provided |
      CashDeskApplicationEventHandlerIf | onCashDeskInitEvent
```

Figure 7: Fragment of error report provided by checking framework for method call

	D5.2: Experiments with impact analysis	
	Version: 1.0	Last change: 29.04.2010

After we fixed the Java implementation to be consistent with the behaviour model in TBP by adding the corresponding method call, the checking framework reported no error.

Last, we also have to check compliance of the TBP behaviour model of the `CashDeskApplication` component with TBP models of other components directly connected to it inside the service architecture, especially the behaviour model of the newly introduced `SecurityManager`, using the Badger tool¹. The Badger tool accepts the TBP models of all subcomponents of a composite component and verifies the compatibility of the communicating component on the level TBP model. More information on the communication compatibility relation can be found in [2]. We ran the check before and after all changes described above were done; in both cases, the Badger tool reported no error.

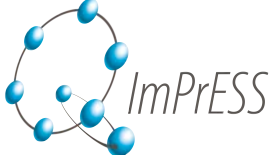
Results of our experiments show that the consistency checking framework can successfully detect inconsistencies between service behaviour model in TBP and implementation in Java that correspond to common evolution scenarios.

5 Conclusion

In this document we described evaluation of the framework for checking consistency between service implementation and its behaviour model in TBP. The results of experiments show that the framework is able to detect inconsistencies that may arise as a consequence of common evolution scenarios described in D1.1.

The main limitation of the framework is that it supports only services implemented in Java. In particular, C and C++ languages are not supported; the reasons are discussed in D5.1 in detail. Another weakness is that the error reports provided by the consistency checking tool may be quite complex in case of really large software systems in which many methods are changed in an evolution step.

¹ At the time of writing this document, the Badger tool is not integrated into the Q-ImPrESS IDE.

	D5.2: Experiments with impact analysis	
	Version: 1.0	Last change: 29.04.2010

6 Glossary

behaviour = control flow and data flow within the components of the static structure

component = A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to third-party composition.

JPF = Java PathFinder, a software model checker tool for Java

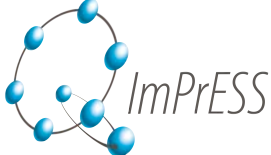
service = a deployed component

service architecture = a set of services connected to each other via connectors. A subject to analysis.

static structure = description of component and connector composition that captures the static architecture of a service at the time of deployment

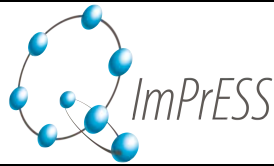
TBP = Threaded Behaviour Protocols, a behaviour model of a component/service specifying traffic on service provided and required interface ports as visible from outside, i.e., by other services communicating with this one

white-box component = a component with available source-code

	D5.2: Experiments with impact analysis	
	Version: 1.0	Last change: 29.04.2010

7 References

- [1] Rausch, A., Reussner, R., Mirandola, R., and Plasil, F.: *The Common Component Modeling Example: Comparing Software Component Models*. 1st. Springer Publishing Company, Incorporated, 2008.
- [2] Kofron, J., Poch, T., Sery, O.: TBP: Code-Oriented Component Behavior Specification, Software Engineering Workshop, Annual IEEE/NASA Goddard, pp. 75-83, 32nd Annual IEEE Software Engineering Workshop, 2008
- [3] Java PathFinder – software model checker, <http://babelfish.arc.nasa.gov/trac/jpf>



8 Appendix

8.1 Modified implementation of the CashDeskApplicationImpl

```
package cocome;

import java.util.List;
import java.util.ArrayList;

import static cocome.CashDeskStates.*;

public class CashDeskApplicationImpl implements CashDeskAppEventHandlerIf
{
    // Required interfaces
    protected BankIf bankIf;
    protected CashDeskConnectorIf cashDeskConnectorIf;
    protected CashDeskEventDispatcherIf cashDeskEventDispatcherIf;
    protected CashDeskApplicationEventDispatcherIf cashDeskAppEventDispatcherIf;
    protected SecurityManagerIf securityManagerIf;

    private CashDeskStates currState = NONE;

    private String cardInformation = null;

    private double runningtotal = 0.0;

    private TransactionID transactionid;

    private StringBuilder total = new StringBuilder("");

    private List<ProductWithStockItemTO> products = new
ArrayList<ProductWithStockItemTO>();

    private boolean expressModeEnabled = false;

    private String cashDeskName;

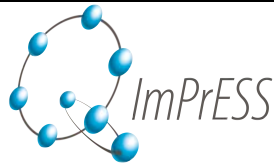
    public CashDeskApplicationImpl()
    {
        cashDeskName = "CashDesk 1"; // +
Integer.toString(Simulator.registerCashDeskApplication(this));
    }

    public void setBankIf(BankIf bank) {
        this.bankIf = bank;
    }

    public void setCashDeskConnectorIf(CashDeskConnectorIf cdc) {
        this.cashDeskConnectorIf = cdc;
    }

    public void setCashDeskEventDispatcherIf(CashDeskEventDispatcherIf cded) {
        this.cashDeskEventDispatcherIf = cded;
    }

    public void
setCashDeskApplicationEventDispatcherIf(CashDeskApplicationEventDispatcherIf cdaed)
    {
```



```
        this.cashDeskAppEventDispatcherIf = cdaed;
    }

    public void setSecurityManagerIf(SecurityManagerIf securityManager) {
        this.securityManagerIf = securityManager;
    }

    // -----
    // Implementation of the CashDeskAppEventHandlerIf interface
    'CashDeskAppEventHandlerIf'
    // -----

    public void onCashDeskInitEvent(CashDeskInitEvent cashDeskInitEvent) {
        System.out.println(cashDeskName + ": Application: CashDeskInitEvent
received");
        if (currState.equals(NONE)) {
            if
(securityManagerIf.checkCredentials(cashDeskInitEvent.getLogin(),
cashDeskInitEvent.getPassword())) {
                currState = INITIALIZED;
            }
        }
    }

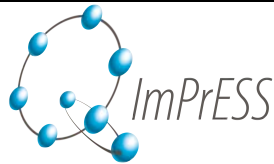
    public void onSaleStartedEvent(SaleStartedEvent saleStartedEvent) {
        System.out.println(cashDeskName + ": Application: SaleStartedEvent
received");
        if (currState.equals(INITIALIZED)) {
            reset();
            currState = SALE_STARTED;
        }
    }

    public void onProductBarcodeScannedEvent(ProductBarcodeScannedEvent
productBarcodeScannedEvent) {
        System.out.println(cashDeskName + ": Application:
ProductBarcodeScannedEvent(" + productBarcodeScannedEvent.getScannedBarcode() + ")
received");
        if (currState.equals(SALE_STARTED)) {
            // stop scanning if more than 8 products in express mode
            if (expressModeEnabled && products.size() == 8) {
                return;
            }
            ProductWithStockItemTO product = null;
            try {
                product =
cashDeskConnectorIf.getProductWithStockItem(productBarcodeScannedEvent.getScannedBa
rcode());
            } catch (NoSuchProductException e) {
                System.out.println(cashDeskName + ": Application: " +
e.toString());
            }

            cashDeskAppEventDispatcherIf.sendProductBarcodeNotValidEvent(new
ProductBarcodeNotValidEvent(productBarcodeScannedEvent.getScannedBarcode()));

            return;
        }

        String productname = product.getName();
        double price = 0;
    }
}
```



```
        if (product.getStockItemTO() != null) price =
product.getStockItemTO().getSalesPrice();

        // calculate running total
        runningtotal += price;
        // round
        runningtotal = Math rint(100 * runningtotal) / 100;

        cashDeskAppEventDispatcherIf.sendRunningTotalChangedEvent(new
RunningTotalChangedEvent(productname, price, runningtotal));

        // if crash here, abort anyway
        products.add(product);
    }
}

    public void onCodeEnteredManuallyEvent(CodeEnteredManuallyEvent
codeEnteredManuallyEvent) {
        System.out.println(cashDeskName + ": Application:
CodeEnteredManuallyEvent(" + codeEnteredManuallyEvent.getEnteredBarcode() + ")
received");
        if (currState.equals(SALE_STARTED)) {
            // stop scanning if more than 8 products in express mode
            if (expressModeEnabled && products.size() == 8) {
                return;
            }
            ProductWithStockItemTO product = null;

            try {
                product = cashDeskConnectorIf
                    .getProductWithStockItem(codeEnteredManuallyEvent
                        .getEnteredBarcode());
            } catch (NoSuchProductException e) {
                System.out.println(cashDeskName + ": Application: " +
e.getMessage());

                cashDeskAppEventDispatcherIf.sendProductBarcodeNotValidEvent(new
ProductBarcodeNotValidEvent(codeEnteredManuallyEvent.getEnteredBarcode()));

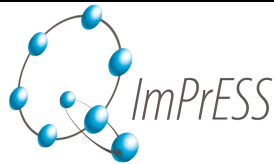
                return;
            }

            String productname = product.getName();
            double price = product.getStockItemTO().getSalesPrice();
            // calculate running total
            runningtotal += price;
            // round
            runningtotal = Math rint(100 * runningtotal) / 100;

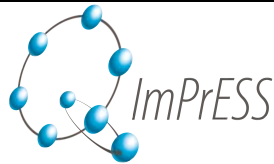
            cashDeskAppEventDispatcherIf.sendRunningTotalChangedEvent(new
RunningTotalChangedEvent(productname, price, runningtotal));

            // if crash here, abort anyway
            products.add(product);
        }
    }

    public void onSaleFinishedEvent(SaleFinishedEvent saleFinishedEvent) {
        System.out.println(cashDeskName + ": Application: SaleFinishedEvent
received");
        if (currState.equals(SALE_STARTED)) {
            currState = SALE_FINISHED;
        }
    }
}
```



```
    }  
}  
  
    public void onCashAmountEnteredEvent(CashAmountEnteredEvent  
cashAmountEnteredEvent) {  
        System.out.println(cashDeskName + ": Application:  
CashAmountEnteredEvent received");  
        if (currState.equals(PAYING_BY_CASH)) {  
            switch (cashAmountEnteredEvent.getKeyStroke()) {  
                case ONE:  
                    total = total.append("1");  
                    break;  
                case TWO:  
                    total = total.append("2");  
                    break;  
                case THREE:  
                    total = total.append("3");  
                    break;  
                case FOUR:  
                    total = total.append("4");  
                    break;  
                case FIVE:  
                    total = total.append("5");  
                    break;  
                case SIX:  
                    total = total.append("6");  
                    break;  
                case SEVEN:  
                    total = total.append("7");  
                    break;  
                case EIGHT:  
                    total = total.append("8");  
                    break;  
                case NINE:  
                    total = total.append("9");  
                    break;  
                case ZERO:  
                    total = total.append("0");  
                    break;  
                case COMMA:  
                    total = total.append(".");  
                    break;  
            }  
        }  
    }  
  
    public void onCashAmountCompletedEvent(CashAmountCompletedEvent  
cashAmountCompletedEvent) {  
        // the ENTER key is pressed at the cash desk keyboard  
        System.out.println(cashDeskName + ": Application:  
CashAmountCompletedEvent received");  
        if (currState.equals(PAYING_BY_CASH)) {  
            try {  
                // prevents NumberFormatException in case no  
                CashAmountEntered event occurred before this one  
                if (total.length() == 0) total = total.append("0");  
  
                double amount = Double.parseDouble(total.toString());  
                double changeamount = amount - runningtotal;  
                // round  
                changeamount = Math rint(100 * changeamount) / 100;  
            }  
        }  
    }  
}
```



```
cashDeskAppEventDispatcherIf.sendChangeAmountCalculatedEvent(new
ChangeAmountCalculatedEvent(changeamount));

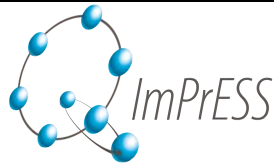
        currState = PAID;
        return;
    } catch (NumberFormatException e) {
        System.out.println("Application: " + e.toString());
    }
}

public void onCashBoxClosedEvent(CashBoxClosedEvent cashBoxClosedEvent) {
    System.out.println(cashDeskName + ": Application: CashBoxClosedEvent
received");
    if (currState.equals(PAID)) {
        makeSale(PaymentMode.CASH);
        reset();
        currState = INITIALIZED;
    }
}

public void onCreditCardScannedEvent(CreditCardScannedEvent
creditCardScannedEvent) {
    System.out.println(cashDeskName + ": Application:
CreditCardScannedEvent received");
    if (currState.equals(PAYING_BY_CREDITCARD)
        || currState.equals(CREDIT_CARD_SCANNED)) {
        cardInformation =
creditCardScannedEvent.getCreditCardInformation();
        currState = CREDIT_CARD_SCANNED;
    }
}

public void onPINEnteredEvent(PINEnteredEvent pinEnteredEvent) {
    System.out.println(cashDeskName + ": Application: PINEnteredEvent
received");
    if (currState.equals(CREDIT_CARD_SCANNED)) {
        transactionid = bankIf.validateCard(cardInformation,
pinEnteredEvent.getPIN());
        if (transactionid == null) {
            System.out.println(cashDeskName + ": Application: issuing
InvalidCreditCardEvent.");
            cashDeskAppEventDispatcherIf.sendInvalidCreditCardEvent(new
InvalidCreditCardEvent());

            return;
        }
        Debit info = bankIf.debitCard(transactionid);
        if (info.equals(Debit.OK)) {
            // make the sale
            System.out.println(cashDeskName + ": Application: Credit
card accepted.");
            makeSale(PaymentMode.CREDITCARD);
            reset();
            currState = INITIALIZED;
        }
        if (info.equals(Debit.TRANSACTION_ID_NOT_VALID)) {
            // we have to rescan the card
            System.out.println(cashDeskName + ": Application: Credit
card payment: Transaction id not valid");
        }
    }
}
```



```
        cashDeskAppEventDispatcherIf.sendInvalidCreditCardEvent(new
InvalidCreditCardEvent());
            currState = PAYING_BY_CREDITCARD;
        }
        if (info.equals(Debit.NOT_ENOUGH_MONEY)) {
            // Other failures
            System.out.println(cashDeskName + ": Application: Credit
card payment: Not enough money on the account");
        }

        cashDeskAppEventDispatcherIf.sendInvalidCreditCardEvent(new
InvalidCreditCardEvent());
    }
}

public void onExpressModeEnabledEvent(ExpressModeEnabledEvent
expressModeEnabledEvent) {
    System.out.println(cashDeskName + ": Application:
ExpressModeEnabledEvent received");
    if (!expressModeEnabled
        /* && expressModeEnabledEvent.getCashdesk().equals(...)*/
    ) {

        cashDeskAppEventDispatcherIf.sendExpressModeEnabledEvent(expressModeEnabledEv
ent);

        // appPublisher.publish(topicSession
//     .createObjectMessage(expressModeEnabledEvent));

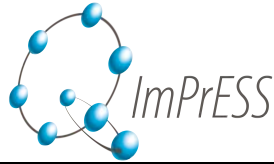
        expressModeEnabled = true;
    }
}

public void onExpressModeDisabledEvent(ExpressModeDisabledEvent
expressModeDisabledEvent) {
    System.out.println(cashDeskName + ": Application:
ExpressModeDisabledEvent received");
    if (expressModeEnabled) {
        expressModeEnabled = false;
    }
}

public void onPaymentModeEvent(PaymentModeEvent paymentModeEvent) {
    System.out.println(cashDeskName + ": Application: PaymentModeEvent
received");
    if (currState.equals(SALE_FINISHED) ||
currState.equals(PAYING_BY_CREDITCARD)) {
        PaymentMode mode = paymentModeEvent.getMode();
        if (mode.equals(PaymentMode.CASH)) {
            currState = PAYING_BY_CASH;
        }
        if (mode.equals(PaymentMode.CREDITCARD) && !expressModeEnabled)
    {
            currState = PAYING_BY_CREDITCARD;
        }
    }
}

// -----
// Private methods
// -----

private void reset() {
```



```
runningtotal = 0.0;
total = new StringBuilder("");
products = new ArrayList<ProductWithStockItemTO>();
cardInformation = null;
}

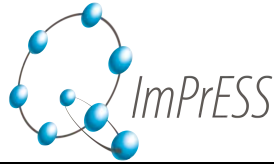
private void makeSale(PaymentMode mode) {
    SaleTO saleTO = new SaleTO();
    saleTO.setProductTOs(products);

    /*
    // the old synchronous call
    CashDeskConnectorIf.bookSale(saleTO);
    */

    cashDeskAppEventDispatcherIf.sendSaleSuccessEvent(new
SaleSuccessEvent());

    // goes to store
    cashDeskEventDispatcherIf.sendAccountSaleEvent(
        new AccountSaleEvent(saleTO)
    );

    // goes to coordinator
    cashDeskEventDispatcherIf.sendSaleRegisteredEvent(
        new SaleRegisteredEvent(
            cashDeskName,
            saleTO.getProductTOs().size(), mode
        )
    );
}
}
```



8.2 Original implementation of the CashDeskApplicationImpl

```
package cocome;

import java.util.List;
import java.util.ArrayList;

import static cocome.CashDeskStates.*;

public class CashDeskApplicationImpl implements CashDeskAppEventHandlerIf
{
    // Required interfaces
    protected BankIf bankIf;
    protected CashDeskConnectorIf cashDeskConnectorIf;
    protected CashDeskEventDispatcherIf cashDeskEventDispatcherIf;
    protected CashDeskApplicationEventDispatcherIf cashDeskAppEventDispatcherIf;

    private CashDeskStates currState = INITIALIZED;

    private String cardInformation = null;

    private double runningtotal = 0.0;

    private TransactionID transactionid;

    private StringBuilder total = new StringBuilder("");

    private List<ProductWithStockItemTO> products = new
ArrayList<ProductWithStockItemTO>();

    private boolean expressModeEnabled = false;

    private String cashDeskName;

    public CashDeskApplicationImpl()
    {
        cashDeskName = "CashDesk 1"; // +
Integer.toString(Simulator.registerCashDeskApplication(this));
    }

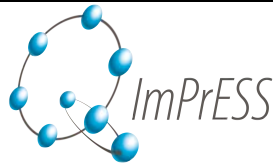
    public void setBankIf(BankIf bank) {
        this.bankIf = bank;
    }

    public void setCashDeskConnectorIf(CashDeskConnectorIf cdc) {
        this.cashDeskConnectorIf = cdc;
    }

    public void setCashDeskEventDispatcherIf(CashDeskEventDispatcherIf cded) {
        this.cashDeskEventDispatcherIf = cded;
    }

    public void
setCashDeskApplicationEventDispatcherIf(CashDeskApplicationEventDispatcherIf cdaed)
    {
        this.cashDeskAppEventDispatcherIf = cdaed;
    }

    // -----
```



```
// Implementation of the CashDeskAppEventHandlerIf interface
'CashDeskAppEventHandlerIf'
// -----

    public void onSaleStartedEvent(SaleStartedEvent saleStartedEvent) {
        System.out.println(cashDeskName + ": Application: SaleStartedEvent
received");
        if (currState.equals(INITIALIZED)) {
            reset();
            currState = SALE_STARTED;
        }
    }

    public void onProductBarcodeScannedEvent(ProductBarcodeScannedEvent
productBarcodeScannedEvent) {
        System.out.println(cashDeskName + ": Application:
ProductBarcodeScannedEvent(" + productBarcodeScannedEvent.getScannedBarcode() + ")
received");
        if (currState.equals(SALE_STARTED)) {
            // stop scanning if more than 8 products in express mode
            if (expressModeEnabled && products.size() == 8) {
                return;
            }
            ProductWithStockItemTO product = null;

            try {
                product =
cashDeskConnectorIf.getProductWithStockItem(productBarcodeScannedEvent.getScannedBa
rcode());
            } catch (NoSuchProductException e) {
                System.out.println(cashDeskName + ": Application: " +
e.toString());

                cashDeskAppEventDispatcherIf.sendProductBarcodeNotValidEvent(new
ProductBarcodeNotValidEvent(productBarcodeScannedEvent.getScannedBarcode()));

                return;
            }

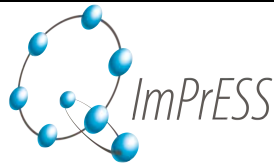
            String productname = product.getName();
            double price = 0;
            if (product.getStockItemTO() != null) price =
product.getStockItemTO().getSalesPrice();

            // calculate running total
            runningtotal += price;
            // round
            runningtotal = Math rint(100 * runningtotal) / 100;

            cashDeskAppEventDispatcherIf.sendRunningTotalChangedEvent(new
RunningTotalChangedEvent(productname, price, runningtotal));

            // if crash here, abort anyway
            products.add(product);
        }
    }

    public void onCodeEnteredManuallyEvent(CodeEnteredManuallyEvent
codeEnteredManuallyEvent) {
        System.out.println(cashDeskName + ": Application:
CodeEnteredManuallyEvent(" + codeEnteredManuallyEvent.getEnteredBarcode() + ")
received");
        if (currState.equals(SALE_STARTED)) {
```



```
// stop scanning if more than 8 products in express mode
if (expressModeEnabled && products.size() == 8) {
    return;
}
ProductWithStockItemTO product = null;

try {
    product = cashDeskConnectorIf
        .getProductWithStockItem(codeEnteredManuallyEvent
            .getEnteredBarcode());
} catch (NoSuchProductException e) {
    System.out.println(cashDeskName + ": Application: " +
e.getMessage());

    cashDeskAppEventDispatcherIf.sendProductBarcodeNotValidEvent(new
ProductBarcodeNotValidEvent(codeEnteredManuallyEvent.getEnteredBarcode()));

    return;
}

String productname = product.getName();
double price = product.getStockItemTO().getSalesPrice();
// calculate running total
runningtotal += price;
// round
runningtotal = Math rint(100 * runningtotal) / 100;

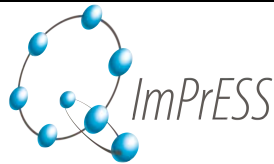
cashDeskAppEventDispatcherIf.sendRunningTotalChangedEvent(new
RunningTotalChangedEvent(productname, price, runningtotal));

// if crash here, abort anyway
products.add(product);
}

}

public void onSaleFinishedEvent(SaleFinishedEvent saleFinishedEvent) {
    System.out.println(cashDeskName + ": Application: SaleFinishedEvent
received");
    if (currState.equals(SALE_STARTED)) {
        currState = SALE_FINISHED;
    }
}

public void onCashAmountEnteredEvent(CashAmountEnteredEvent
cashAmountEnteredEvent) {
    System.out.println(cashDeskName + ": Application:
CashAmountEnteredEvent received");
    if (currState.equals(PAYING_BY_CASH)) {
        switch (cashAmountEnteredEvent.getKeyStroke()) {
            case ONE:
                total = total.append("1");
                break;
            case TWO:
                total = total.append("2");
                break;
            case THREE:
                total = total.append("3");
                break;
            case FOUR:
                total = total.append("4");
                break;
            case FIVE:
```



```
        total = total.append("5");
        break;
    case SIX:
        total = total.append("6");
        break;
    case SEVEN:
        total = total.append("7");
        break;
    case EIGHT:
        total = total.append("8");
        break;
    case NINE:
        total = total.append("9");
        break;
    case ZERO:
        total = total.append("0");
        break;
    case COMMA:
        total = total.append(".");
        break;
    }
}

}

    public void onCashAmountCompletedEvent(CashAmountCompletedEvent
cashAmountCompletedEvent) {
    // the ENTER key is pressed at the cash desk keyboard
    System.out.println(cashDeskName + ": Application:
CashAmountCompletedEvent received");
    if (currState.equals(PAYING_BY_CASH)) {
        try {
            // prevents NumberFormatException in case no
CashAmountEntered event occurred before this one
            if (total.length() == 0) total = total.append("0");

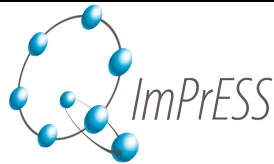
            double amount = Double.parseDouble(total.toString());
            double changeamount = amount - runningtotal;
            // round
            changeamount = Math rint(100 * changeamount) / 100;

            cashDeskAppEventDispatcherIf.sendChangeAmountCalculatedEvent(new
ChangeAmountCalculatedEvent(changeamount));

            currState = PAID;
            return;
        } catch (NumberFormatException e) {
            System.out.println("Application: " + e.toString());
        }
    }
}

    public void onCashBoxClosedEvent(CashBoxClosedEvent cashBoxClosedEvent) {
    System.out.println(cashDeskName + ": Application: CashBoxClosedEvent
received");
    if (currState.equals(PAID)) {
        makeSale(PaymentMode.CASH);
        reset();
        currState = INITIALIZED;
    }
}

    public void onCreditCardScannedEvent(CreditCardScannedEvent
creditCardScannedEvent) {
```



```
        System.out.println(cashDeskName + ": Application:
CreditCardScannedEvent received");
        if (currState.equals(PAYING_BY_CREDITCARD)
            || currState.equals(CREDIT_CARD_SCANNED)) {
            cardInformation =
creditCardScannedEvent.getCreditCardInformation();
            currState = CREDIT_CARD_SCANNED;
        }
    }

    public void onPINEnteredEvent(PINEnteredEvent pinEnteredEvent) {
        System.out.println(cashDeskName + ": Application: PINEnteredEvent
received");
        if (currState.equals(CREDIT_CARD_SCANNED)) {
            transactionid = bankIf.validateCard(cardInformation,
                pinEnteredEvent.getPIN());
            if (transactionid == null) {
                System.out.println(cashDeskName + ": Application: issuing
InvalidCreditCardEvent.");

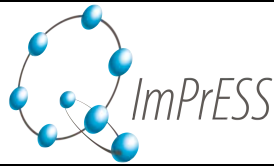
                cashDeskAppEventDispatcherIf.sendInvalidCreditCardEvent(new
InvalidCreditCardEvent());

                return;
            }
            Debit info = bankIf.debitCard(transactionid);
            if (info.equals(Debit.OK)) {
                // make the sale
                System.out.println(cashDeskName + ": Application: Credit
card accepted.");
                makeSale(PaymentMode.CREDITCARD);
                reset();
                currState = INITIALIZED;
            }
            if (info.equals(Debit.TRANSACTION_ID_NOT_VALID)) {
                // we have to rescan the card
                System.out.println(cashDeskName + ": Application: Credit
card payment: Transaction id not valid");

                cashDeskAppEventDispatcherIf.sendInvalidCreditCardEvent(new
InvalidCreditCardEvent());
                currState = PAYING_BY_CREDITCARD;
            }
            if (info.equals(Debit.NOT_ENOUGH_MONEY)) {
                // Other failures
                System.out.println(cashDeskName + ": Application: Credit
card payment: Not enough money on the account");

                cashDeskAppEventDispatcherIf.sendInvalidCreditCardEvent(new
InvalidCreditCardEvent());
            }
        }
    }

    public void onExpressModeEnabledEvent(ExpressModeEnabledEvent
expressModeEnabledEvent) {
        System.out.println(cashDeskName + ": Application:
ExpressModeEnabledEvent received");
        if (!expressModeEnabled
            /* && expressModeEnabledEvent.getCashdesk().equals(...)*/
        ) {
```



```
cashDeskAppEventDispatcherIf.sendExpressModeEnabledEvent (expressModeEnabledEvent);
    // appPublisher.publish(topicSession
    //     .createObjectMessage (expressModeEnabledEvent));
    expressModeEnabled = true;
}
}

public void onExpressModeDisabledEvent (ExpressModeDisabledEvent
expressModeDisabledEvent) {
    System.out.println(cashDeskName + ": Application:
ExpressModeDisabledEvent received");
    if (expressModeEnabled) {
        expressModeEnabled = false;
    }
}

public void onPaymentModeEvent (PaymentModeEvent paymentModeEvent) {
    System.out.println(cashDeskName + ": Application: PaymentModeEvent
received");
    if (currState.equals (SALE_FINISHED) ||
currState.equals (PAYING_BY_CREDITCARD)) {
        PaymentMode mode = paymentModeEvent.getMode();
        if (mode.equals (PaymentMode.CASH)) {
            currState = PAYING_BY_CASH;
        }
        if (mode.equals (PaymentMode.CREDITCARD) && !expressModeEnabled)
        {
            currState = PAYING_BY_CREDITCARD;
        }
    }
}

// -----
// Private methods
// -----

private void reset() {
    runningtotal = 0.0;
    total = new StringBuilder("");
    products = new ArrayList<ProductWithStockItemTO>();
    cardInformation = null;
}

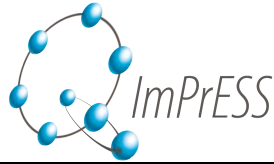
private void makeSale (PaymentMode mode) {
    SaleTO saleTO = new SaleTO();
    saleTO.setProductTOs (products);

    /*
    // the old synchronous call
    CashDeskConnectorIf.bookSale (saleTO);
    */

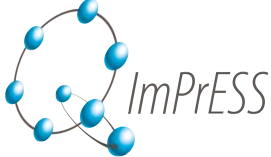
    cashDeskAppEventDispatcherIf.sendSaleSuccessEvent (new
SaleSuccessEvent ());

    // goes to store
    cashDeskEventDispatcherIf.sendAccountSaleEvent (
        new AccountSaleEvent (saleTO)
    );

    // goes to coordinator
```

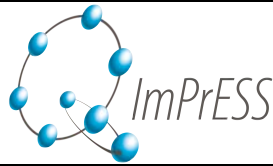


```
cashDeskEventDispatcherIf.sendSaleRegisteredEvent(  
    new SaleRegisteredEvent(  
        cashDeskName,  
        saleTO.getProductTOs().size(), mode  
    )  
);  
}  
}
```

	D5.2: Experiments with impact analysis	
	Version: 1.0	Last change: 29.04.2010

8.3 Source code of SecurityManagerIf interface

```
package cocome;  
  
public interface SecurityManagerIf {  
    boolean checkCredentials(String login, String password);  
}
```



8.4 Modified implementation of CashDeskAppEventHandlerIf

```
package cocome;
```

```
public interface CashDeskAppEventHandlerIf {

    public void onCashDeskInitEvent(CashDeskInitEvent cashDeskInitEvent);

    /**
     * Event handler for SaleStartedEvent events.
     */
    public void onSaleStartedEvent(SaleStartedEvent saleStartedEvent);

    /**
     * Event handler for ProductBarcodeScannedEvent events.
     */
    public void onProductBarcodeScannedEvent(ProductBarcodeScannedEvent
productBarcodeScannedEvent);

    /**
     * Event handler for CodeEnteredManuallyEvent events.
     */
    public void onCodeEnteredManuallyEvent(CodeEnteredManuallyEvent
codeEnteredManuallyEvent);

    /**
     * Event handler for SaleFinishedEvent events.
     */
    public void onSaleFinishedEvent(SaleFinishedEvent saleFinishedEvent);

    /**
     * Event handler for CashAmountEnteredEvent events.
     */
    public void onCashAmountEnteredEvent(CashAmountEnteredEvent
moneyAmountEnteredEvent);

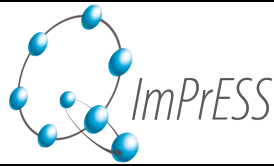
    /**
     * Event handler for CashAmountCompletedEvent events.
     */
    public void onCashAmountCompletedEvent(CashAmountCompletedEvent
cashAmountCompletedEvent);

    /**
     * Event handler for CashBoxClosedEvent events.
     */
    public void onCashBoxClosedEvent(CashBoxClosedEvent cashBoxClosedEvent);

    /**
     * Event handler for CreditCardScannedEvent events.
     */
    public void onCreditCardScannedEvent(CreditCardScannedEvent
creditCardScannedEvent);

    /**
     * Event handler for PINEnteredEvent events.
     */
    public void onPINEnteredEvent(PINEnteredEvent pinEnteredEvent);

    /**
     * Event handler for ExpressModeEnabledEvent events.
     */
}
```



```
public void onExpressModeEnabledEvent(ExpressModeEnabledEvent
expressModeEnabledEvent);

/**
 * Event handler for ExpressModeEnabledEvent events.
 */
public void onExpressModeDisabledEvent(ExpressModeDisabledEvent
expressModeDisabledEvent);

/**
 * Event handler for PaymentModeEvent events.
 */
public void onPaymentModeEvent(PaymentModeEvent paymentModeEvent);
}
```

8.5 Original implementation of CashDeskAppEventHandlerIf

```
package cocome;
```

```
public interface CashDeskAppEventHandlerIf {

/**
 * Event handler for SaleStartedEvent events.
 */
public void onSaleStartedEvent(SaleStartedEvent saleStartedEvent);

/**
 * Event handler for ProductBarcodeScannedEvent events.
 */
public void onProductBarcodeScannedEvent(ProductBarcodeScannedEvent
productBarcodeScannedEvent);

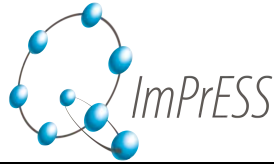
/**
 * Event handler for CodeEnteredManuallyEvent events.
 */
public void onCodeEnteredManuallyEvent(CodeEnteredManuallyEvent
codeEnteredManuallyEvent);

/**
 * Event handler for SaleFinishedEvent events.
 */
public void onSaleFinishedEvent(SaleFinishedEvent saleFinishedEvent);

/**
 * Event handler for CashAmountEnteredEvent events.
 */
public void onCashAmountEnteredEvent(CashAmountEnteredEvent
moneyAmountEnteredEvent);

/**
 * Event handler for CashAmountCompletedEvent events.
 */
public void onCashAmountCompletedEvent(CashAmountCompletedEvent
cashAmountCompletedEvent);

/**
 * Event handler for CashBoxClosedEvent events.
 */
public void onCashBoxClosedEvent(CashBoxClosedEvent cashBoxClosedEvent);
}
```



```
/**
 * Event handler for CreditCardScannedEvent events.
 */
public void onCreditCardScannedEvent (CreditCardScannedEvent
creditCardScannedEvent);

/**
 * Event handler for PINEnteredEvent events.
 */
public void onPINEnteredEvent (PINEnteredEvent pinEnteredEvent);

/**
 * Event handler for ExpressModeEnabledEvent events.
 */
public void onExpressModeEnabledEvent (ExpressModeEnabledEvent
expressModeEnabledEvent);

/**
 * Event handler for ExpressModeDisabledEvent events.
 */
public void onExpressModeDisabledEvent (ExpressModeDisabledEvent
expressModeDisabledEvent);

/**
 * Event handler for PaymentModeEvent events.
 */
public void onPaymentModeEvent (PaymentModeEvent paymentModeEvent);
}
```