UNIVERSITY OF OSLO
Department of Informatics

INF5150
Obligatory Exercise
Drop 2

Group 3:
Bjørn Olav Samdal,
Florian Müller,
Ingar Vindenes,
Shawn Svendsen,
Tom Lysemose

18th November 2005

Contents

1 Introduction **4**
- 1.1 Rational Software Modeler . . . 4

2 Executable UML Design **5**
- 2.1 Class Diagram . . . 5
- 2.2 Signals . . . 6
- 2.3 Composite Structure: MBDS . . . 7
- 2.4 Composite Structure: Controller . . . 8
- 2.5 Composite Structure: EventHandler . . . 10
- 2.6 Controller State Machine . . . 12
- 2.7 Session State Machine . . . 14
- 2.8 EventHandler State Machine . . . 16
- 2.9 Event State Machine . . . 18
- 2.10 User explanation . . . 20
- 2.11 Using the GUI . . . 20
- 2.12 Using SMS . . . 21
- 2.13 Additional information . . . 21

3 Risk Analysis **22**
- 3.1 Context identification . . . 22
- 3.2 Risk identification . . . 25
- 3.3 Risk analysis . . . 26
- 3.4 Risk evaluation . . . 27
- 3.5 Risk treatment . . . 28

4 Refinement proof **30**

1 Introduction

This paper describes our proposal for a solution to the second obligatory exercise in INF5150, or Drop2. The main goal in Drop2 is to create an executable UML design of the solution provided by the teachers to the first exercise - hereafter called Drop1[1].In section 2 on the facing page we will go into detail about our design, and how to run and test it.

The provided solution, that our solution is built on, is set of models describing a system called "*"Multiple Blind Date System"* (MBDS). As the name suggest, it is a system to help people meet others - without knowing in advance who they will meet. This is done by letting the users of the system send SMS messages to join events. Before the event take place the MBDS-service will return a SMS message containing information about the event location, and how to get there to the user. Drop1 consist mainly of as set of assumptions, a class diagram, the composite structure and sequence diagrams modelling the MBDS-service. The diagrams show how three different specifications can be implemented, and in addition how the last two are refinements of the first.

After the executable UML design, we will briefly explain how we may understand the design as a refinement of Drop1.

The next part of our solution is a risk analysis using the CORAS tool. The CORAS tool is a system built on open standards that support the CORAS methodology. It is a efficient way of performing a risk analysis in a structured approach. The requirements limit the analysis to four risks that does not meet the risk evaluation criteria - to limit the work load.

1.1 Rational Software Modeler

The main tool used to create the executable UML design is the Rational Software Modeler (RSM). The UML design will be compiled into JavaFrame through a RSM-plugin provided by the course management. The installation and adjustments to make this work took quite a long time.

The RSM software used is not the perfect tool to make models, especially regarding the state machines. We also had problems when exporting the diagrams as images. Sometimes RSM didn't include all the parts of the diagram when exporting it as an image - without giving any error messages - making the exported diagram faulty.

[1] This should not be mixed with our proposal for the first obligatory exercise, which was also called Drop1

2 Executable UML Design

2.1 Class Diagram

The class diagram is equal to the class diagram of the solution of Drop1. The phone numbers[2] of the customers are stored as Strings. When a list as a data type was needed, the class `ArrayList` out of the `java.util` package was used. The classes `LocationSupplier`, `Event` and `ControllerSM` contain attributes of this data type.

The class `PhoneNo` just contains the attribute `number` of the type `String`.

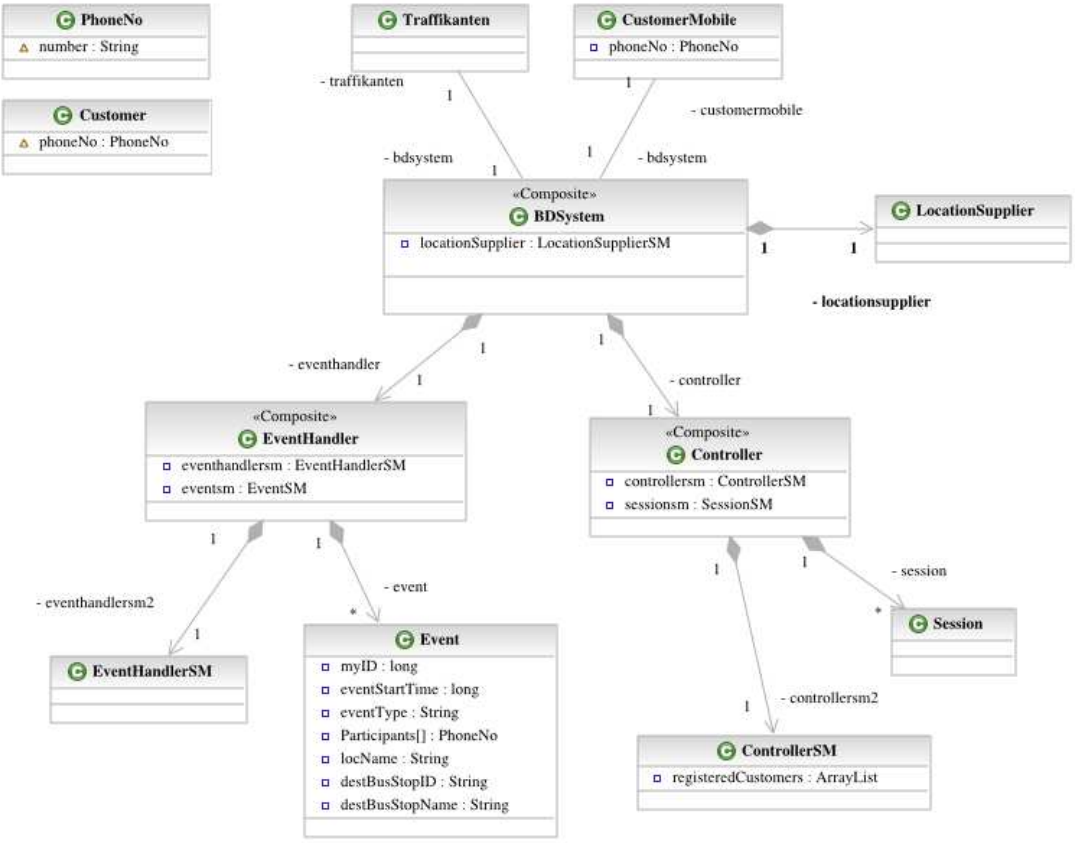

Figure 1: MBDS class diagram

[2] For the unique identification of the users a unique sms identifier, given by PATS, is used.

2.2 Signals

The following signals are used within our system:

- **ClosestBusStop** (PhoneNo phoneNo, String depBusStopID, String depBusStopName)
- **EventMade** (String eventType, long time, String locName)
- **GetClosestBusStop** (PhoneNo phoneNo, String custPos)
- **GetLocation** (PhoneNo phoneNo)
- **GetLocInfo** (String locName, long eventID)
- **JoinEvent** (PhoneNo phoneNo, String eventType, long time)
- **JoinEventOk** (PhoneNo phone)
- **LocInfo** (long eventID, String destBusStopID, String destBusStopName)
- **MakeEvent** (String eventType, long time, String locName)
- **TooMany** (PhoneNo phoneNo)
- **TrigEventMessage** (PhoneNo phoneNo, String eventType, String locName, String destBusStopID, String destBusStopName)

2.3 Composite Structure: MBDS

Since it is a given constraint by JavaFrame, parts within the composite structure diagrams can only be either another composite or a statemachine. Our composite `BDSystem` equals the `BDSystem`-composite of the Drop1 with a additional `InputEdgeMediator` called *"Gui-InputMediator"* This mediator offers a graphical user interface for a simplified usage. The other mediators use the `DynOutMediator` and `SmsOutputMediator` types.

The parts `controller` and `eventhandler` are other composites, `locationsupplier` is a statemachine, which will be desribed in a more more detailed later on.

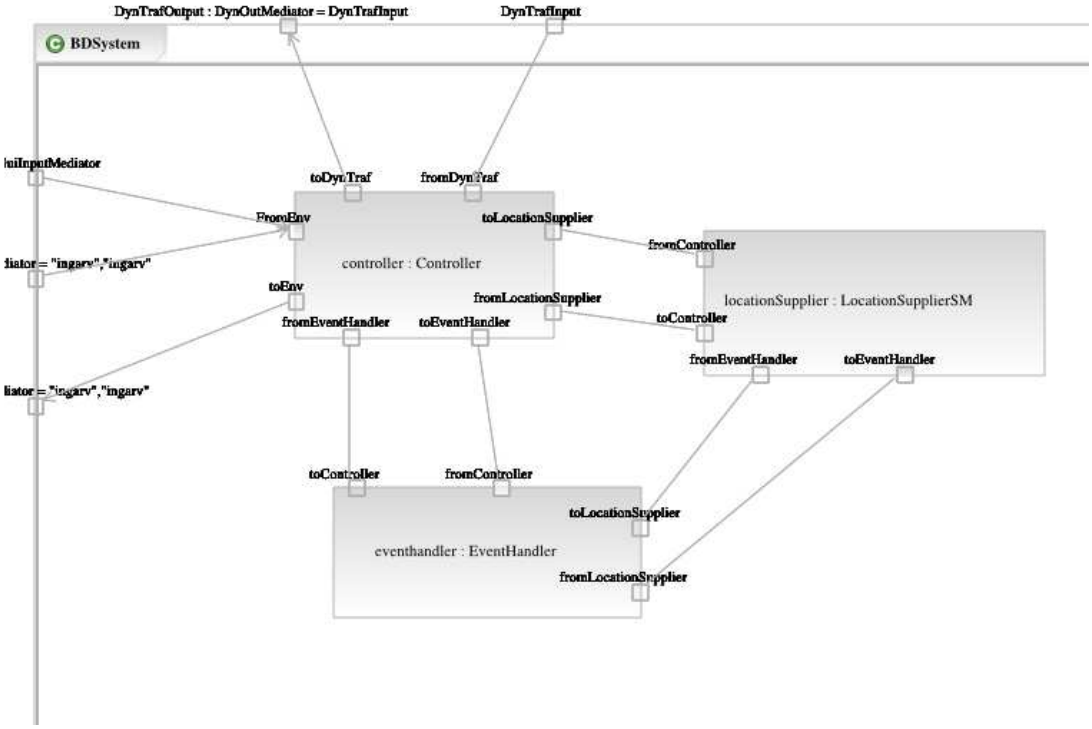

Figure 2: Composite Structure MBDS

2.4 Composite Structure: Controller

The `Controller` composite is also very similar to the given composite of Drop1. It contains two parts:

1. `controllersm` of type `ControllerSM`: It is a statemachine und has a central role, it handles all incoming sms messages and forwards incoming messages from the location supplier and the eventhandler to the right session instances.
2. `sessionsm` of type `SessionSM`: A instance of the type Session is created everytime a customer wants to join an event. "It handles the communication with the customer related to his particular event" (from Drop1 solution).

If the controller receives a SMS message, the following preprocessing algorithm will be executed within the SMSEffect2 activity:

Listing 1: SmsEffect2 activity

```
System.out.println("[ControllerSM] SMS received: "+sig.getMessage());
//preprocess sms message text
sig.setMessage(sig.getMessage().substring(sig.getMessage().indexOf(" ") ↩
    +1));
sig.setMessage(sig.getMessage().substring(sig.getMessage().indexOf(" ") ↩
    +1));
sig.setMessage(sig.getMessage().substring(sig.getMessage().indexOf(" ") ↩
    +1));
System.out.println("[ControllerSM] Preprocessed message text: "+sig. ↩
    getMessage());
```

As one can see, `ControllerSM` contains the mediator `toSession` which type is `DynSessionRouter`. This class has the stereotype `SimpleRouterMediator` and forwards messages to dynamical created instances of the class `Session` with the following algorithm:

Listing 2: DynSessionRouter activity

```
// forward to requested session
String phonenumber = new String();
if( sig instanceof JoinEventOk ) {
   phonenumber = ((JoinEventOk)sig).phone.number;
} else if( sig instanceof TrigEventMessage ) {
   phonenumber = ((TrigEventMessage)sig).phoneNo.number;
} else if( sig instanceof PosResult ) {
   System.out.println("[ControllerSM] Received PosResult from "+(( ↩
       PosResult)sig).getMessageId());
   phonenumber = ((PosResult)sig).getMessageId();
} else if( sig instanceof ClosestBusStop ) {
   phonenumber = ((ClosestBusStop)sig).phoneNo.number;
} else if( sig instanceof DynInfo ) {
   phonenumber = ((DynInfo)sig).getRoutingInfo();
} else if( sig instanceof TooMany ) {
   phonenumber = ((TooMany)sig).phoneNo.number;
```

```
}

for (int i=0; i<mediatorList.size(); i++) {
   ToSessionSMMediator mediator = (ToSessionSMMediator) mediatorList.get( ↩
       i);

   if (phonenumber.equals(mediator.phoneNo.number) ) {
      mediator.forward(sig);
      System.out.println("[DynSessionRouter] DynRouter: sig is forwarded" ↩
          );
   }
}
```

Instances of the type `SessionSM` got the parameters `phoneNo`, `eventType` and `time` with the data types `PhoneNo`, `String` and `long`. More details about the usage of these parameters will be given later on.

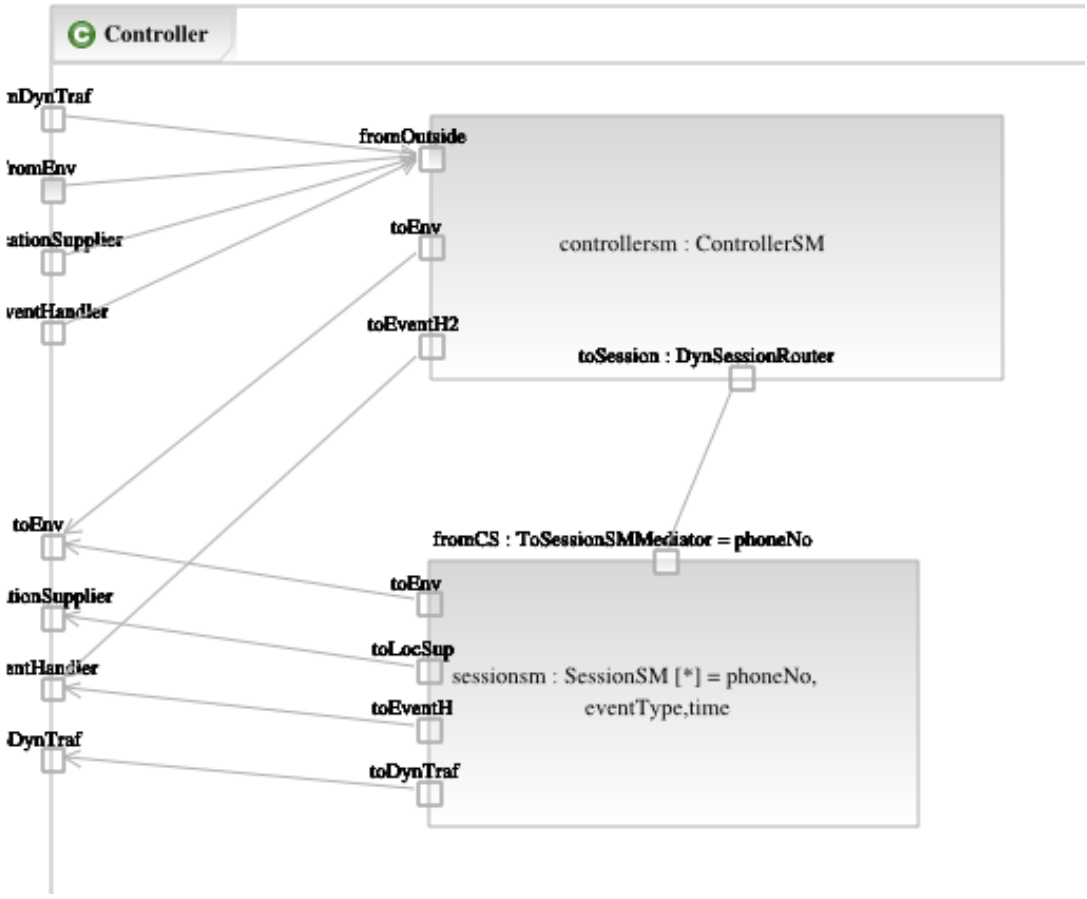

Figure 3: Composite Structure: Controller

2.5 Composite Structure: EventHandler

The composite `EventHandler` contains the parts `eventhandlersm` of the type `EventHandlerSM` and dynamical created parts `event` type `Event`. `eventhandlersm` creates new events and forwards messages that are addressed to single events. Instances of the type `Event` are dynamical created by `eventhandler` as response on a `MakeEvent` signal. They expect the parameters `eventStartTime` (`long`), `eventType` (`String`), `locName` (`String`) and `myID` (`long`), which is a unique identifier for the event.

The `eventhandlersm` forwards messages for single events through the port `toEvent` which is of the type `DynEventRouter`. `DynEventRouter` is a class with the stereotype `SimpleRouterMediator` and uses the following algorithm for the message forwarding:

Listing 3: DynEventRouter activity

```
if( sig instanceof LocInfo ) {
   // forward message to requested event
   LocInfo locinfo = (LocInfo)sig;
   for (int i=0; i<mediatorList.size(); i++) {
   ToEventSMMediator mediator = (ToEventSMMediator) mediatorList.get(i);

   System.out.println("[DynEventRouter] DynRouter: mediator.myID = " + ↩
       mediator.myID);
   System.out.println("[DynEventRouter] DynRouter: locinfo.eventID = " + ↩
       locinfo.eventID);

   if (locinfo.eventID==mediator.myID ) {
      mediator.forward(locinfo);

      System.out.println("[DynEventRouter] DynRouter: locinfo is ↩
          forwarded");
      }
   }
} else if( sig instanceof JoinEvent ) {
   // forward message to requested event
   JoinEvent joinevent = (JoinEvent)sig;
   for (int i=0; i<mediatorList.size(); i++) {
   ToEventSMMediator mediator = (ToEventSMMediator) mediatorList.get(i);

   if (joinevent.time==mediator.time && joinevent.eventType. ↩
       equalsIgnoreCase(mediator.eventType) ) {
      mediator.forward(joinevent);

      System.out.println("[DynEventRouter] DynRouter: joinevent is ↩
          forwarded");
      }
   }
}
```

Figure 4: Composite Structure: EventHandler

2.6 Controller State Machine

The `ControllerSM` receives incoming SMS messages. It checks, wether it is a join message or another one[3]. If a SMS message is recognized as a join message, a new instance of the type `SessionSM` is created. Otherwise the following algorithm processes the SMS message:

Listing 4: SmsEffect activity

```
// Handles SMS messages
// sms format: to, from, message
// message format: join-eventType-time OR make-eventType-time-locName OR register-me
csm.user_id=sig.getFrom();
csm.smstext=sig.getMessage();
String[] parts = csm.smstext.split("-");
System.out.println("Received message: user_id: "+csm.user_id+", message parts ("+parts.length+"):");
for( int i=0; i<parts.length; i++ ) {
   System.out.print(parts[i]+" ");
}
System.out.println();
if( parts[0].equalsIgnoreCase("make") ) { // make-message
   // make event-handling
   System.out.println("[ControllerSM] Create event");
   output(new MakeEvent(parts[1],Long.parseLong(parts[2]),parts[3]), csm.toEventH2,csm);
} else if( parts[0].equalsIgnoreCase("register") ) {  // register message
   if(csm.registeredCustomers==null ) { csm.registeredCustomers=new ArrayList(); }
   csm.registeredCustomers.add(new Customer(new PhoneNo(csm.user_id)));
   System.out.println("[ControllerSM] Register customer "+csm.user_id+". Over all: "+csm.registeredCustomers.size());
   output(new Sms("Successfully registered "+csm.user_id,csm.user_id,"2034"),csm.toEnv,csm);
}
```

If an event was created by the event handler, the controller will notify all registered customers about the new event. This happens, when the controller receives a message of the type EventMade and is shown in the following algorithm:

Listing 5: EventMade activity

```
// Notify all registered users about new event
EventMade em=(EventMade)sig;
if( csm.registeredCustomers!=null ) {
   ListIterator li=csm.registeredCustomers.listIterator();
   while( li.hasNext() ) {
      Customer c=(Customer)li.next();
      System.out.println("[ControllerSM] Notifying "+c.phoneNo.number+" about new event");
      output(new Sms("New event: Type: "+em.eventType+", Loc: "+em.locName+" at "+em.time,c.phoneNo.number,"2034"),csm.toEnv,csm);
   }
}
```

[3] In our implementation the SMS message types are *join*, *register* and *make*.

All other incoming messages are forwarded to the single sessions whose identifiers are contained within the single messages.

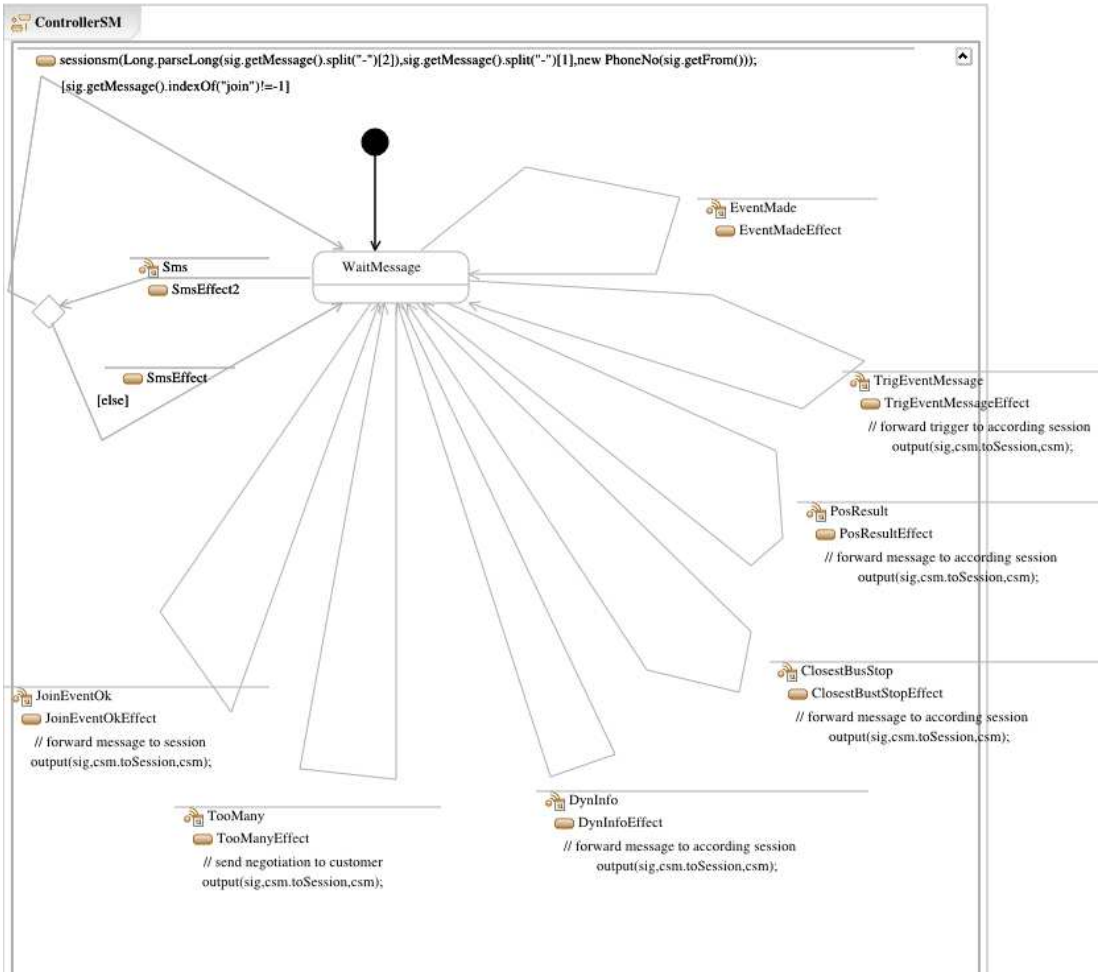

Figure 5: Controller State Machine

2.7 Session State Machine

The SessionSM handles the communication with a single customer related to a particular event.

When a SessionSM is created, it sends a JoinEvent message to the event handler that is supposed to forward this message to the single event:

Listing 6: JoinEventEffect activity of SessionSM

```
// send join event message to event handler
output(new JoinEvent(csm.phoneNo,csm.eventType,csm.time),csm.toEventH,csm );
```

Afterwards, the instance of SessionSM is waiting for a reply of the event. If the event is full, we will receive a TooMany message and send a notification about this to the customer, saying that the joining is not possible:

Listing 7: TooManyEffect activity

```
// send negotiation to customer
output(new Sms("Sorry, but you cannot join the event "+csm.eventType+" at "+csm.time+", because it is too full.",csm.phoneNo.number,"2034"), csm.toEnv,csm);
```

If the joining is possible, we will receive a JoinEventOk message from the event and we will send a confirmation for the successful joining to the customer:

Listing 8: JoinEventOkEffect activity

```
// send confirmation to customer
output(new Sms("Successfully joined the event "+csm.eventType+" at "+csm.time+" !",csm.phoneNo.number,"2034"),csm.toEnv,csm);
```

In the state *"WaitingForEventTrigger"* the session is waiting for the notification trigger. At an appropriate time, the SessionSM instance will get a TrigEventMessage message, which contains information about the event that is going to take place. These information will be stored and the location of the customers mobile will be requested at PATS:

Listing 9: trigEventMessageEffect activity

```
// save event information
csm.locName=sig.locName;
csm.destBusStopID=sig.destBusStopID;
csm.destBusStopName=sig.destBusStopName;
System.out.println("[SessionSM] Session "+csm.phoneNo.number+" was triggered: Event at "+csm.locName);

// get Location of the customer
PosRequest pr=new PosRequest(csm.phoneNo.number);
pr.setMessageId(csm.phoneNo.number);
output(pr,csm.toEnv,csm);
```

After we received the position, we store it and ask for the closest bus stop regarding this position:

Listing 10: PosResultEffect activity

```
// get closest Bus stop
csm.smstext = sig.getPositioningResult();

int ix = csm.smstext.indexOf("<Breddegrad>");
csm.latitude = csm.smstext.substring(ix+12,ix+19);
ix = csm.smstext.indexOf("<Lengdegrad>");
csm.longitude = csm.smstext.substring(ix+12,ix+20);
System.out.println("[SessionSM] Tracked position of customer "+csm.
    phoneNo.number+": "+csm.latitude+" "+csm.longitude);

output(new GetClosestBusStop(csm.phoneNo,csm.latitude+" "+csm.longitude),
    csm.toLocSup,csm);
```

After we received the closest bus stop, we have all information that we need for a route request to Trafikanten:

Listing 11: ClosestBusStopEffect activity

```
// save results
csm.depBusStopID=sig.depBusStopID;
csm.depBusStopName=sig.depBusStopName;
System.out.println("[SessionSM-"+csm.phoneNo.number+"] Stored departure
    information: "+csm.depBusStopName+" ("+csm.depBusStopID+")");

// DynRequest to Trafikanten
String dynroute="SN$"+csm.depBusStopID;
System.out.println("[SessionSM-"+csm.phoneNo.number+"] DynRoute request:
    "+dynroute);
output(new DynRequest(dynroute,csm.phoneNo.number),csm.toDynTraf,csm);
```

In the end we send a notification message to the customer:

Listing 12: DynInfoEffect activity

```
// send notification SMS to customer
System.out.println("[SessionSM-"+csm.phoneNo.number+"] Received DynRoute
    message");

DynRoute dr=((DynInfo)sig).getDynRoutes()[0];
// send notification to customer
System.out.println("[SessionSM] Notification is sent to customer "+csm.
    phoneNo.number);
output(new Sms(csm.eventType+" at "+csm.time+": Take line #"+ dr.
    getLineText()+" from "+csm.depBusStopName+" to "+dr.
    getDestinationStop()+" at "+dr.getExpectedDepatureTime().substring
    (11,16)+". Get out at: "+csm.destBusStopName, csm.phoneNo.number, "
    2034"), csm.toEnv, csm);
```

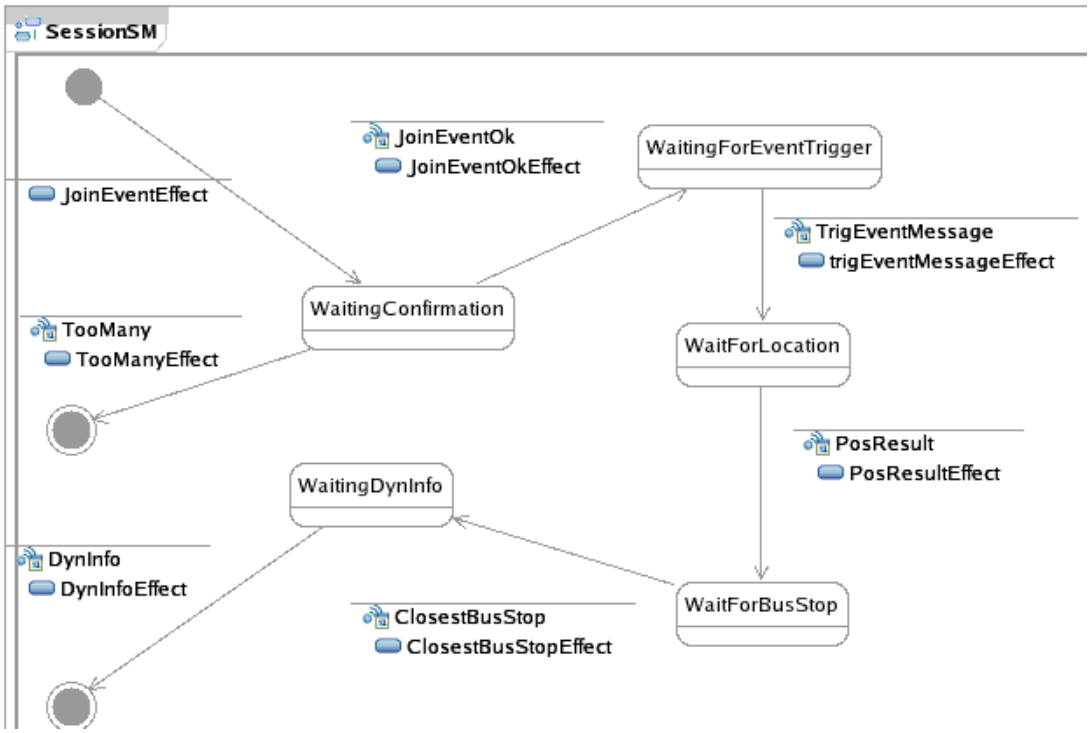

Figure 6: Session State Machine

2.8 EventHandler State Machine

The event handler has basically two functions:

1. **Create a new event** - When a `MakeEvent` message is received the event handler creates a new instance of `Event`. The unique identifier of the the single events is a static counter of the type `long` that is incremented everytime before a new instance is created.
2. **Forward messages to single events** - Both, `JoinEvent` and `LocInfo` messages, are forwarded to the single events by the `DynEventRouter` port of `eventhandlersm`.

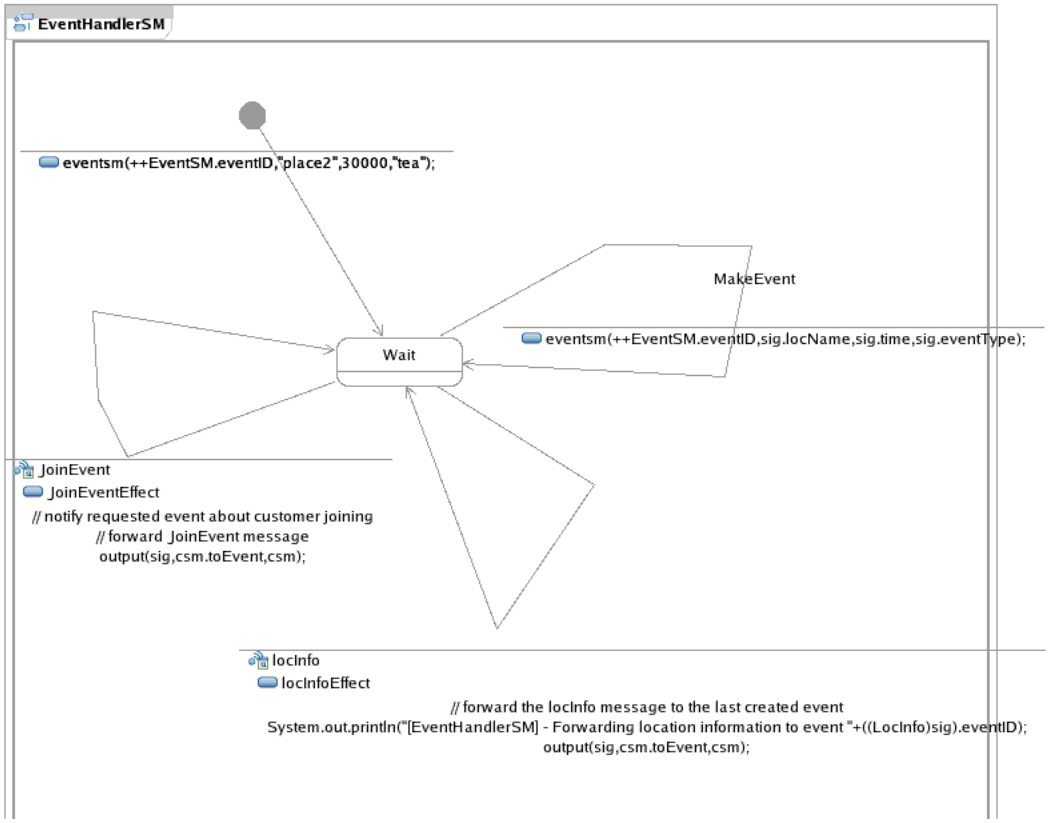

Figure 7: EventHandler State Machine

2.9 Event State Machine

Instances of the `EventSM` will be created every time, a new event is created, either by the vendor itself or by a customer. Right after the creation, some variables, e.g. for storing the participating customers and the timer, are initialized and the closest bus stop for the location of the event is requested at the:

Listing 13: InitializeEffect of ControllerSM

```
// Initialize participants array
csm.participants = new ArrayList();
// set maximal number of participants - here for test purposes only 2
csm.maxparts=2;

// Initialize timer
csm.countdown.setDelay((int)csm.eventStartTime);
csm.countdown.startTimer();

System.out.println("[EventSM] EventSM "+csm.myID+" created");
System.out.println("[EventSM] Requesting location information for event " +csm.myID);
output(new GetLocInfo(csm.locName,csm.myID),csm.toLocationSup,csm);
```

After we received the information about the closest bus stop of the location of the event, the event sends a `EventMade` message to the controller:

Listing 14: LocInfoEffect activity

```
// receive location information
LocInfo locinfo=(LocInfo)sig;
csm.destBusStopID=locinfo.destBusStopID;
csm.destBusStopName=locinfo.destBusStopName;
// notify Controller about new event
output(new EventMade(csm.eventType,csm.eventStartTime,csm.locName),csm. toController,csm);
```

In the *"WaitPersons"* state we are actually waiting for persons to join the event. If we receive a `JoinEvent` message, we check, if there is a seat left. If there is a seat left, we add the customer and send a `JoinEventOk` message to the controller, otherwise we send a `TooMany` message to the controller:

Listing 15: JoinEventEffect activity

```
// add customer to event
JoinEvent je = (JoinEvent)sig;
if( csm.participants.size()<csm.maxparts ) {
   csm.participants.add(je.phoneNo);
   System.out.println("[EventSM] Customer "+je.phoneNo.number+" added to "+csm.eventType+" at "+csm.eventStartTime);
   // notify controller
   output(new JoinEventOk(je.phoneNo),csm.toController,csm);
} else {
   System.out.println("[EventSM] Customer "+je.phoneNo.number+" CANNOT be added to "+csm.eventType+" at "+csm.eventStartTime);
```

```
10    // notify controller
      output(new JoinEventOk(je.phoneNo),csm.toController,csm);
}
```

When the timer of the event is finished, a `TrigEventMessage` for each of the customers that joined the event is sent to the controller. This message contains all information which are important for the further notification of the single customers.

Listing 16: TimerEffect

```
// send trigger event messages for each participant
ListIterator li=csm.participants.listIterator();
while( li.hasNext() ) {
4     PhoneNo pn = (PhoneNo)li.next();
      output(new TrigEventMessage(pn,csm.eventType,csm.locName,csm. ↩
          destBusStopID,csm.destBusStopName) ,csm.toController,csm);
}
```

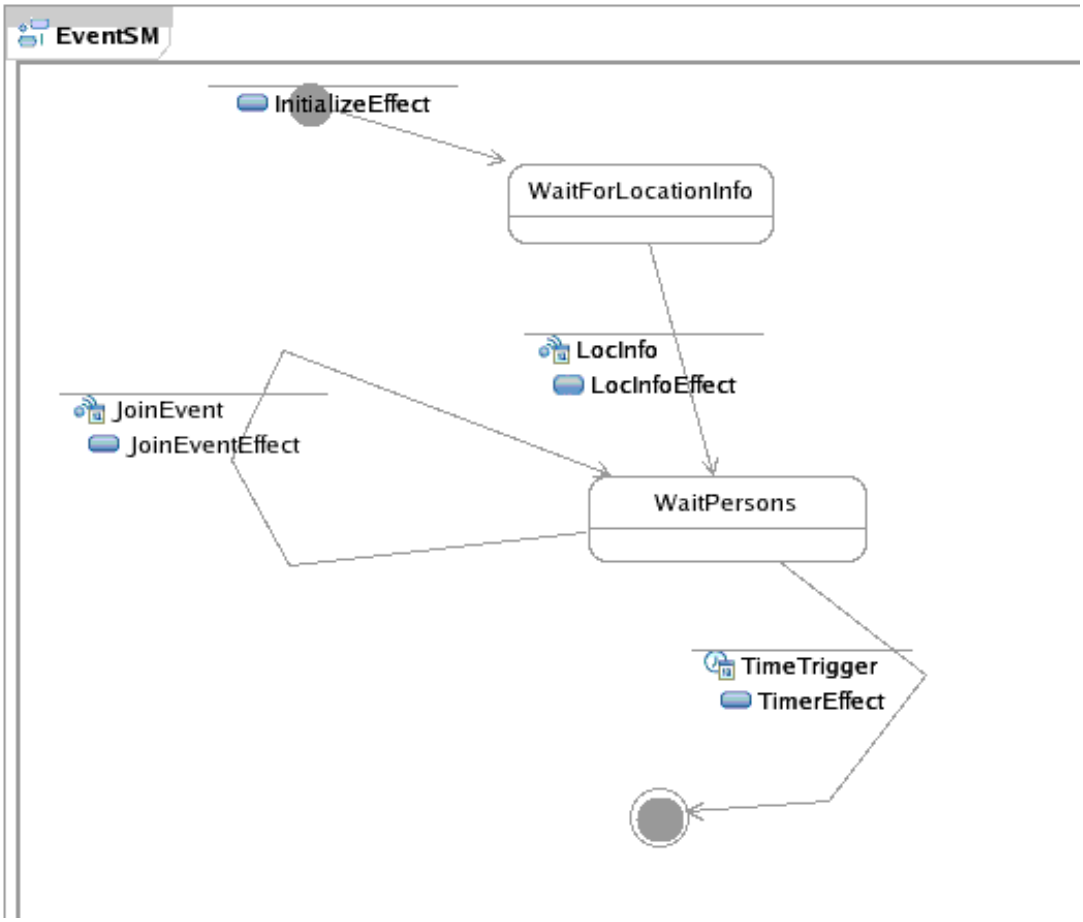

Figure 8: Event State Machine

2.10 User explanation

A test event will be created right after starting up the system. It will be triggered 30 seconds after starting the system. The eventType is “tea”, the eventTime is 30000.

There are information for three locations stored in the system: place1, place2 and place3. This means that all other event locations, that are created during the use of the system will have the central station as a recommend meeting place (see description of the l ocastion supplier for further details).

The system can be used on two ways:

1. Using the GUI
2. Using SMS

There are three different types of messages that the user can send to the Blind Date System:

1. Join message
2. Register message
3. Make event message

Parameters that have to be changed by the user are written italic.

2.11 Using the GUI

Make event message The make event message has the following structure: “stud1 konto *username* make-*eventType-eventTime-eventLocation*,2034,*SMSID*”. Note that eventTime in this implementation is the time in milliseconds that has to pass by until the event will be triggered. It is important to use “-” as a separator - the algorithm within the system splits the message with the help of this sign. SMSID is provided by PATS. E.g. “stud1 konto florianm make-tea-60000-place3,2034,YourSmsIdHere”

Join message To join an event, send a message with the following structure: “stud1 konto *username* join-*eventType-eventTime*,2034,*SMSID*”. username has to be replaced by a real username. Note that eventType and eventTime are used as unique identifiers for the single events. This means that these must be exactly the same like in the ones in the system. E.g. “stud1 konto florianm join-tea-30000,2034,YourSmsIdHere” within the first 30 seconds after the start up of the system.

Register message The register message has the following structure: “stud1 konto *username* register-me,2034,*SMSID*”.

2.12 Using SMS

All SMSs have to be sent to the number 2034. Only Telenor mobiles can be used.

Join message Send a SMS with the following structure to 2034: stud1 konto *username* join-*eventType-eventTime*. E.g. “stud1 konto florianm join-tea-30000” within the first 30 seconds after the start up of the system.

Register message Send a SMS with the following structure to 2034: stud1 konto *username* register-me

Make event message Send a SMS with the following structure to 2034: stud1 konto username make-*eventType-eventTime-eventLocation*. E.g. “stud1 konto florianm make-tea-60000-place3”

2.13 Additional information

The system, as it is modelled and implemented, does not check if a customer has already joined an event. The instances of `SessionSM`, which handle the communication with the customer related to a particular event, use the unique SMS identifier to distinguish between the single instances. By testing our system, we came to the following result: Due to the lack of a controll mechanism for a ”multiple joining” of a single customer, the system will send multiple `JoinEventOk` messages to a single `SessionSM` instance, causing a transition error, which can also be seen with JFTrace.

3 Risk Analysis

3.1 Context identification

Target of Evaluation Table

Type: Table
Name: Target of Evaluation Table
Short description: What parts of the drop that is being evaluated
Concern: Target of evaluation
Viewpoint:
Finalised:
Full description: This ToE is meant to give a analysis of the INF5150 Obligatory Exercise Drop 2, Autumn 2005. This analysis is based on the assistants' solution to Drop 1.

Table 1: Target Of Evaluation Table

Category	Value
Target	Blinddatesystem - A multible blind date service based on SMS messages, that will handle participants linked to one or more events.
Client	The client is the "owners" of the BlindDateSystem.
Service/Function	- Register new client - Manage join request - Provide travelingroute to user - Positiong of meetingplace and user
Quality aspects	- Data confidentiality regarding user data - Availability of system

Value Definition Table

Type: Table
Name: Value Definition Table
Short description:
Concern: Target of evaluation
Viewpoint:
Finalised:
Full description:

Table 2: Value Definition Table

Type	Domain	Allowed values	Description
Asset		Low, Medium, High	
Frequency		Rare, Unlikely, Possible, Likely, Certain	
Consequence		Insignificant, Minor, Moderate, Major	
Risk value		Low, Moderate, Major	

Risk Definition Matrix

Type: Table
Name: Risk Definition Matrix
Short description: A matrix for evaluation risk values
Concern: Target of evaluation
Viewpoint:
Finalised:
Full description: This matrix defines the risk value depending on the frequency and consequence of occurance.

Table 3: Risk Matrix

Frequency	Insignificant	Minor	Moderate	Major
Rare	Low	Low	Low	Low
Unlikely	Low	Low	Low	Moderate
Possible	Low	Low	Moderate	Major
Likely	Moderate	Moderate	Major	Major
Certain	Major	Major	Major	Major

Risk Criteria Evaluation Matrix

Type: Table
Name: Risk Criteria Evaluation Matrix
Short description: Matrix showing where the RiskID is situated
Concern: Target of evaluation
Viewpoint:
Finalised:
Full description: The matrix shows where the different risks are situated in a matrix of consequence and frequency. The matrix will show how the risks evaluete according to the Risk Evaluation Criteria Table.

Table 4: Risk Matrix

Frequency	Insignificant	Minor	Moderate	Major
Rare				R-5
Unlikely				R-6, R-7, R-8, R-9, R-10
Possible			R-11, R-12	R-3, R-4, R-13
Likely		R-2, R-14	R-1	
Certain				

Asset Table

Type: Table
Name: Asset Table
Short description: Clients assets
Concern: Assets
Viewpoint:
Finalised:
Full description: A table of assets for the client at the time og analysis. a4 (User) is set to low, due to the lack of users at the moment of analysis.

Table 5: Asset Table

Asset ID	Description	Category	Value
A-reputation	BDS Reputation	Organisational	Medium
A-trust	Customers Trust	Human	Medium
A-personaldata	Users personal data	Information	High
A-user	Current users	Human	Low
A-systemdesign	Systemdesign	Software	High
A-equipment	Data equipment	Physical	Medium

Risk Evaluation Criteria Table

Type: Table
Name: Risk Evaluation Criteria Table
Short description: Criterias for the different assets
Concern: Risk evaluation criteria
Viewpoint:
Finalised:
Full description: The goal of this activity is to identify the risk evaluation criteria, i.e. what loss in asset value the client can tolerate over a given time interval.

Table 6: Risk Evaluation Criteria Table

Criteria ID	Criteria	Description	Applied for assets
C-1	Accepted if lower or equal to moderate	Default value for criteria.	A-reputation, A-trust, A-personaldata, A-user, A-systemdesign, A-equipment

3.2 Risk identification

HazOp Table

Type: Table
Name: HazOp Table
Short description: Hazard and Operability Analysis
Concern: Threats
Viewpoint:
Finalised:
Full description:

Table 7: HazOp Table

HazOp ID	Asset ID	Reference	Guideword	Incident	Scenario
H-1	A-reputation	sd MakeEvent	spaming	User is spammed by eventnotification service.	Users creates multible new events that is broadcasted to all users over a short time period.
H-2	A-user	sd MakeEvent	spaming	User is spammed by eventnotification service.	Users creates multible new events that is broadcasted to all users over a short time period.
H-3	A-trust	sd MakeEvent	misuse	Users become offended by eventnotifications.	An usercreated event contains a eventdescription that breaks with providers policy. The voilation is broadcasted.
H-4	A-reputation	sd MakeEvent	misuse	The provider is assosiated with an eventpolicy voilation content.	An usercreated event contains a eventdescription that breaks with providers policy. The voilation is broadcasted.
H-5	A-equipment	BDS Hardware/data	Theft	Hardware is stolen, data is lost.	A breakin at the space where the BD System is located
H-6	A-systemdesign	BDS Hardware	Theft	System design get into hands of competitor.	A breakin at the space where the BD System is located
H-7	A-personaldata	BDS Hardware/Data	Theft	Stolen data is published or misused.	Personal data is stolen
H-8	A-trust	BDS	Theft	Stolen data is published or misused.	Personal data is stolen
H-9	A-user	BDS	Theft	Stolen data is published or misused.	Personal data is stolen
H-10	A-user	sd NotifyCustomers	Availibility	User gets no transportroute, and no eventlocation.	The BDS is uable to get the required data back from Trafikanten.
H-11	A-user	sd NotifyCustomer	Timeschedul e	User misses the suggested transport.	After reciving transportroute, the nearest busstop can not be reached in time.
H-12	A-reputation	BDS Hardware	Availibility	BDS is down	Power-shortage occurs.
H-13	A-user	BDS Hardware	Availibility	BDS is down	Power-shortage occurs.
H-14	A-trust	BD_JoinEvent	Delay	Users is added multible times in the event.	Slow response when handling a joiningrequest makes user resend joinrequest.

3.3 Risk analysis

Consequnce and Frequency Table

Type: Table
Name: Consequnce and Frequency Table
Short description:
Concern: Consequence
Viewpoint:
Finalised:
Full description: The goal of this activity is to analyse, evaluate and document the consequence of the unwanted incidents, and frequency evaluation is to come up with a realistic estimate for the probability that each specific unwanted incident occurs.

Table 8: Consequence and Frequency Table

Risk ID	Asset ID	Incident	Consequence Value	Frequency Value
R-1	A-reputation	User is spammed by eventnotification service.	Moderate	Likely
R-2	A-user	User is spammed by eventnotification service.	Minor	Likely
R-3	A-user	Users become offended by eventnotifications.	Major	Possible
R-4	A-reputation	The provider is assosiated with an eventpolicy voilation content.	Major	Possible
R-5	A-equipment	Hardware is stolen, data is lost.	Minor	Rare
R-6	A-systemdesign	System design get into hands of competitor.	Major	Unlikely
R-7	A-personaldata	Stolen data is published or misused.	Major	Unlikely
R-8	A-trust	Stolen data is published or misused.	Major	Unlikely
R-9	A-user	Stolen data is published or misused.	Major	Unlikely
R-10	A-user	User gets no transportroute, and no eventlocation.	Major	Unlikely
R-11	A-user	User misses the suggested transport.	Moderate	Possible
R-12	A-reputation	BDS is down	Moderate	Possible
R-13	A-user	BDS is down	Major	Possible
R-14	A-trust	Users is added multible times in the event.	Minor	Likely

3.4 Risk evaluation

Risk Evaluation Table

Type: Table
Name: Risk Evaluation Table
Short description: Table for showing the priority of risks
Concern: Risk estimates
Viewpoint:
Finalised:
Full description: Each risk not fullfilling the Risk Evaluation Criteria Table is assined a priority. All other risks are assigned a 0.

Table 9: Risk Evaluation Table

Risk ID	Risk Value	Risk Priority
R-3	Major	1
R-4	Major	1
R-1	Major	2
R-13	Major	3
R-2	Moderate	0
R-6	Moderate	0
R-7	Moderate	0
R-8	Moderate	0
R-9	Moderate	0
R-10	Moderate	0
R-11	Moderate	0
R-12	Moderate	0
R-14	Moderate	0
R-5	Low	0

3.5 Risk treatment

Treatment Table

Type: Table
Name: Treatment Table
Short description: Proposed Treatments
Concern: Treatment
Viewpoint:
Finalised:
Full description: For the Risk that did not satisfy the Risk Evaluation Criteria Table, treatment is proposed in this table.

Table 10: Treatment Identification Table

Treatment ID	Risk ID/category	Treatment strategy	Description	References
T-1	R-3	Avoid	Human revision of incoming eventproposals	EventCreationRevision_Treatment (T-3, T-4)
T-2	R-4	Avoid	Human revision of incoming eventproposals	EventCreationRevision_Treatment (T-3, T-4)
T-3	R-1	Reduce frequency	Setting av limit to the number of event offers sent over a periode of time. Limiting number of usercreated event per user.	Spamming_Treatment Diagram (T-1)
T-4	R-13	Reduce frequency	By buying and installing an UPS (Uninterruptible Power Supply) on the server, a loss of power does not cause the server to go down due to shorter power-shortages.	UPS_Treatment (T-4)

UPS_Treatment (T-4)

Type:	UML Model
Name:	UPS_Treatment (T-4)
Short description:	Treatmentdiagram for T-4
Concern:	Treatment
Viewpoint:	
Finalised:	
Full description:	By buying and installing an UPS (Uninterruptible Power Supply) on the server, a loss of power does not cause the server to go down due to shorter power-shortages.

Figure 1: UPS_Treatment (T-4)

EventCreationRevision_Treatment (T-3, T-4)

Type:	UML Model
Name:	EventCreationRevision_Treatment (T-3, T-4)
Short description:	Treatmentdiagram for T-3 & T-4
Concern:	Treatment
Viewpoint:	
Finalised:	
Full description:	An usercreated event contains a eventdescription that breaks with providers policy. The voilation is broadcasted.

Figure 2: EventCreationRevision_Treatment (T-3, T-4)

Spamming_TreatmentDiagram (T-1)

Type:	UML Model
Name:	Spamming_TreatmentDiagram (T-1)
Short description:	Treatmentdiagram for T-1
Concern:	Treatment
Viewpoint:	
Finalised:	
Full description:	Users creates multible new events that is broadcasted to all users over a short time period.

Figure 3: Spamming_TreatmentDiagram (T-1)

4 Refinement proof

In this section we argue that our design may be understood as a refinement of the Drop1 specification. Since our design does not contain any sequence-diagrams, we must argue that our state machines satisfy the Drop1 specification. To do this we must show two things:

1. That our state-machines do not implement any negative trace.
2. That at least one positive trace is implemented (i.e. that we have in fact an implementation of the specification).

Since the Drop1 specification does not define any negative traces, the first proof is trivial. Hence, in a trivial sense of refinement any state-machine will be a implementation of the Drop1 specification (by making a inconclusive trace positive, which is narrowing). In a less trivial notion of refinement, one may demand that the state-machines are in fact an implementation of the specification, by demanding that at least one positive trace is implemented. This is proven by the below trace, produced by our system.

Time	State Machine	Current State	Input	Transition Behaviour	Next State
0	New EventHandlerSM@454...				
0	New ControllerSM@6a19c8f5				
0	New LocationSupplierSM@...				
2583	EventHandlerSM@454c48f5	null	StartMessage@5f4f08f5	New EventSM@26df48f5	Wait
2613	ControllerSM@6a19c8f5	null	StartMessage@6a2b08f5		WaitMessage
2623	LocationSupplierSM@68b5...	null	StartMessage@687908f5		Wait
3224	EventSM@26df48f5	null	StartMessage@213588f5		WaitForLocationInfo
3224	LocationSupplierSM@68b5...	Wait	GetLocInfo@5c9288f6 (place2, 1)	Output LocInfo@575f08f6 (1, 3010624, Oslo gate)	Wait
3355	EventHandlerSM@454c48f5	Wait	LocInfo@575f08f6 (1, 3010624, Oslo gate)	Output LocInfo@575f08f6 (1, 3010624, Oslo gate)	Wait
3425	EventSM@26df48f5	WaitForLocationInfo	ClosestBusStop@1db688f5 (Package1.PhoneNo@23ce0	Output EventMade@6a0e48f6 (tea,	WaitPersons
3475	ControllerSM@6a19c8f5	WaitMessage	EventMade@6a0e48f6 (tea, 30000, place2)		WaitMessage
8242	ControllerSM@6a19c8f5	WaitMessage	Sms@65b8c8f5 (2034,A-AFOCMO,stud1 konto florianm	New SessionSM@342e08f5	WaitMessage
8252	SessionSM@342e08f5	null	StartMessage@356ec8f5	Output JoinEvent@4a4388f5 (Package1.PhoneNo@23ce0 8f5, tea, 30000)	WaitingConfirmation
8252	EventHandlerSM@454c48f5	Wait	JoinEvent@4a4388f5 (Package1.PhoneNo@23ce0 8f5, tea, 30000)	Output JoinEvent@4a4388f5 (Package1.PhoneNo@23ce0 8f5, tea, 30000)	Wait
8252	EventSM@26df48f5	WaitPersons	JoinEvent@4a4388f5 (Package1.PhoneNo@23ce0 8f5, tea, 30000)	Output JoinEventOk@4dc9c8f5 (Package1.PhoneNo@23ce0 8f5)	WaitPersons
8252	ControllerSM@6a19c8f5	WaitMessage	JoinEventOk@4dc9c8f5 (Package1.PhoneNo@23ce0 8f5)	Output JoinEventOk@4dc9c8f5 (Package1.PhoneNo@23ce0 8f5)	WaitMessage
8252	SessionSM@342e08f5	WaitingConfirmation	JoinEventOk@4dc9c8f5 (Package1.PhoneNo@23ce0 8f5)	Output Sms@430848f5 (A-AFOCMO,2034,Successfully joined the event tea at 30000 !)	WaitingForEventTrigger
33228	EventSM@26df48f5	WaitPersons	TimerMsg@2583c8f5	Output TrigEventMessage@564088f5 (Package1.PhoneNo@23ce0 8f5, tea, place2, 3010624, Oslo gate)	FinalState
33228	ControllerSM@6a19c8f5	WaitMessage	TrigEventMessage@564088f5 (Package1.PhoneNo@23ce0 8f5, tea, place2, 3010624, Oslo gate)	Output TrigEventMessage@564088f5 (Package1.PhoneNo@23ce0 8f5, tea, place2, 3010624, Oslo gate)	WaitMessage
33228	SessionSM@342e08f5	WaitingForEventTrigger	TrigEventMessage@564088f5 (Package1.PhoneNo@23ce0 8f5, tea, place2, 3010624, Oslo gate)	Output PosRequest@68db48f5	WaitForLocation
38185	ControllerSM@6a19c8f5	WaitMessage	PosResult@223148f5	Output PosResult@223148f5	WaitMessage
38215	SessionSM@342e08f5	WaitForLocation	PosResult@223148f5	Output GetClosestBusStop@264bc8f5 (Package1.PhoneNo@23ce0 8f5, N595751 E0104400)	WaitForBusStop
38225	LocationSupplierSM@68b5...	Wait	GetClosestBusStop@264bc8f5 (Package1.PhoneNo@23ce0 8f5, N595751 E0104400)	Output ClosestBusStop@1db688f5 (Package1.PhoneNo@23ce0 8f5, 3010345, Lindern)	Wait
38495	ControllerSM@6a19c8f5	WaitMessage	ClosestBusStop@1db688f5 (Package1.PhoneNo@23ce0 8f5, 3010345, Lindern)	Output ClosestBusStop@1db688f5 (Package1.PhoneNo@23ce0 8f5, 3010345, Lindern)	WaitMessage
38565	SessionSM@342e08f5	WaitForBusStop	ClosestBusStop@1db688f5 (Package1.PhoneNo@23ce0 8f5, 3010345, Lindern)	Output DynRequest@2de8c8f5 (SN$3010345)	WaitingDynInfo
40278	ControllerSM@6a19c8f5	WaitMessage	DynInfo@244548f6 (2005-11-17T16:03:06.886+01:00, ok, 0)(37:205:2, Helsfyr T, 37)	Output DynInfo@244548f6 (2005-11-17T16:03:06.886+01:00, ok, 0)(37:205:2, Helsfyr T, 37)	WaitMessage
40338	SessionSM@342e08f5	WaitingDynInfo	DynInfo@244548f6 (2005-11-17T16:03:06.886+01:00, ok, 0)(37:205:2, Helsfyr T, 37)	Output Sms@449308f6 (A-AFOCMO,2034,tea at 30000: Take line #37 from Lindern to Helsfyr T at 16:45. Get out at: Oslo gate)	FinalState

Figure 9: Complete Positive Trace