Solution to drop 1
INF5150, Autumn 2005

Submitted by project group 1:
Stoltenberg Vibeke
Koudrik Igor
Robøle Torunn
Thorvaldsen Kjersti

Table of contents

Table of figures

1. Introduction

We are going to make a system called Multiple Blind Date System (MBDS) which will assist in setting up a meeting for a group of people at a place that is optimally suited for those who will join. We assume that we are a third party vendor that offers a number of events identified by their type and time, e.g. "Go for a milkshake at 8 PM Saturday." To join such an event the user sends an SMS message to MBDS's dedicated number. At an appropriate time before the event, our system will send a message to the participant how she/he should use public transportation to reach the given meeting place. We assume that the mobile phones sending the want-to-join-message can be positioned, and that we may use Trafikanten system to find the nearest bus stop, bus schedule and actual delays.

2. Assumptions

Our system operates on the following assumptions:

- All events are created before the system is put into operation by the system administrator. It is only one object per event. After the scheduled event has expired, the EventHandle resets it. That means that a new time “CurrentTimeOfEvent” is set. The participants are also deleted.

- The system does not check whether a person has joined events that overlap.

- We assume that an event does not belong to a group of events, but is set with both type and place independently.

- Mobile phone from where a participant sends want-to-join-message can be positioned--the participant is assumed to have the same mobile phone number as was used for registration for an event.

- A participant can register for an event not later than one hour before the start of the event. One hour before the event starts it closes for registration, and the system finds the optimal place for the participants. This is system's time interval.

3. MBDS class diagram

The system is thought to have the following classes: EventHandler, Event, Participant, PositionService and Trafikanten. EventHandler manages instances of class Event. CurrentTimeOfEvent, Occurence and PrepareStartOfEvent is used to decide when the Event should happen (time trigging). Instances of class Event manage instances of class Participant. Instances of class Event use Trafikanten class services to get the route, whereas instances of class Participant use PositionService services to find coordinates of participant mobiletelephone.

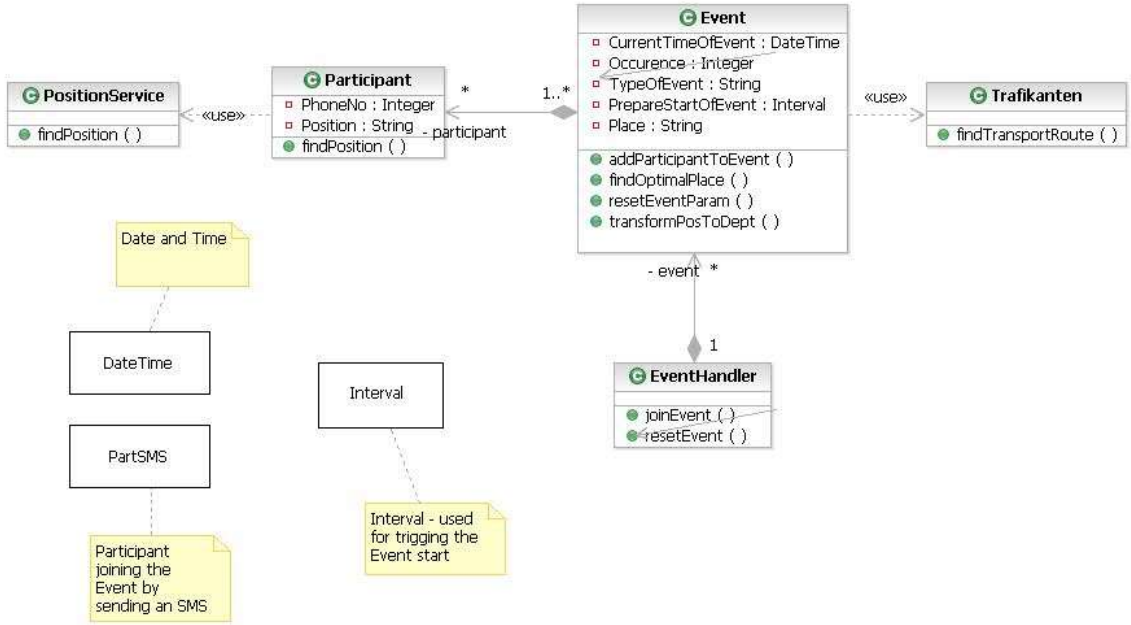

Figure 1 MBDS class diagram.

4. MBDS composite structure diagram

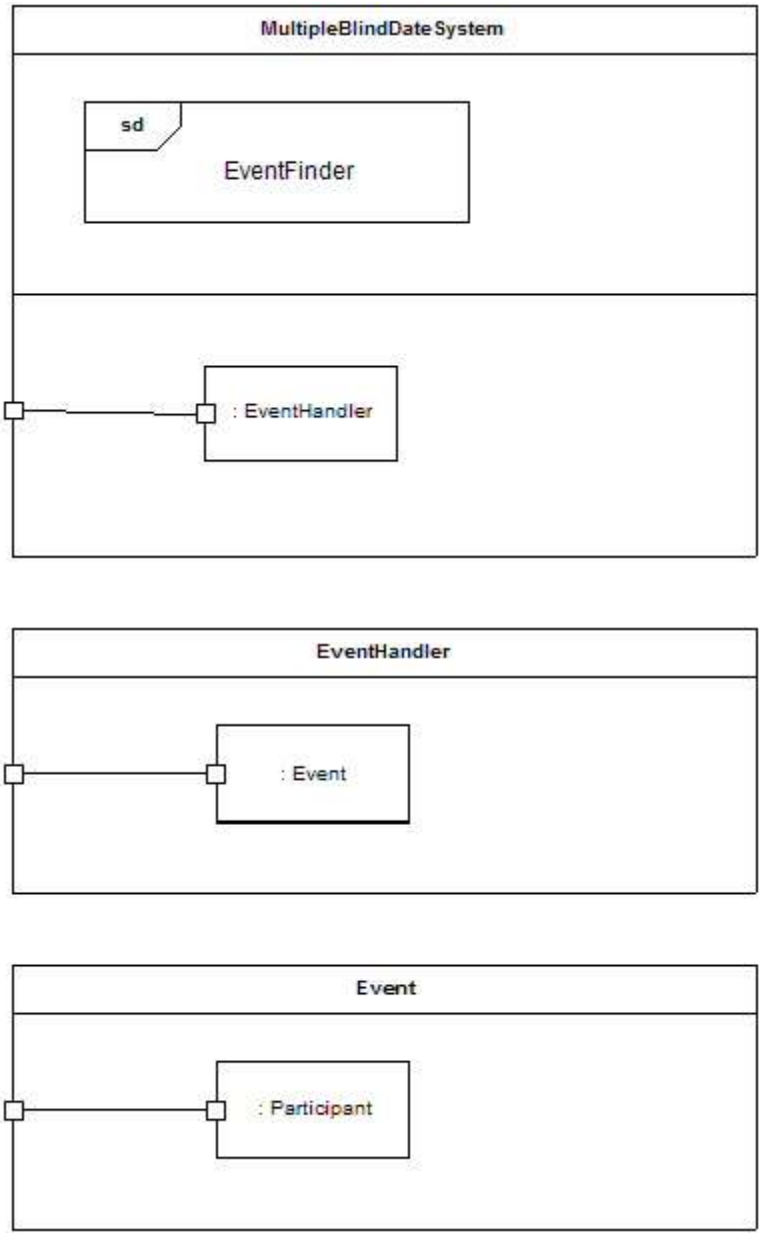

Figure 2 MBDS composite structure diaram.

5. MBDS sequence diagrams

We have three sequence diagrams: EventFinderAbs, EventFinderExt and EventFinderErr.

5.1. SD: EventFinder_Abs

The sequence diagram EventFinder_Abs—the most abstract representation of the system--describes how the MBDS works. The system knows the place for a given Event. The participant joins an Event by sending an SMS. The system uses participant's telephone number to find out his/her coordinates. After that the coordinates are transformed into a concrete spot. This data is sent to the Trafikanten-system and is used together with destination (place for the event) and date-time to find bus schedule and eventual delays. This information is brought back to the participant who sent request to join the event.

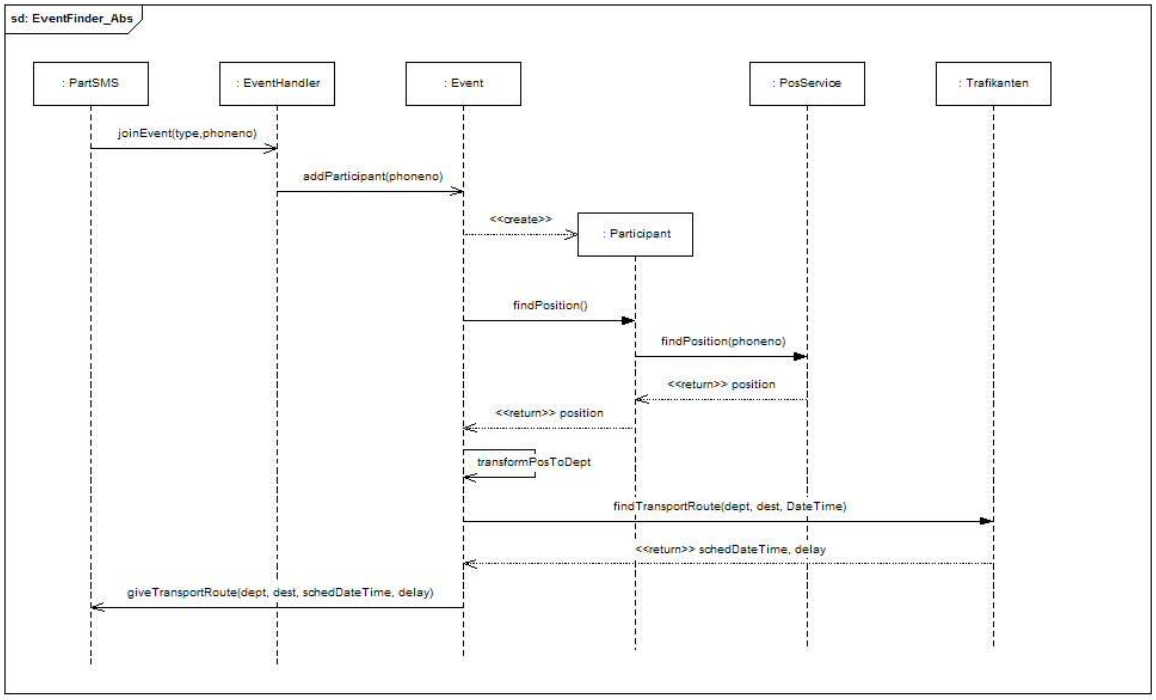

Figure 3 SD: EventFinder_Abs.

5.2. SD: EventFinder_Ext

EventFinder_Ext is an extention of EventFinder_Abs. It specifies time constraint, which is defined as Time = CurrentTimeOfEvent – 1 hour. The time constraint functions as a guard, which negates the process of joining an Event if the condition of the time constraint is not met. EventFinder_Ext also finds an optimal meting place for participants of an event, instead of having a predefined meeting place as it is done in EventFinder_Abs.

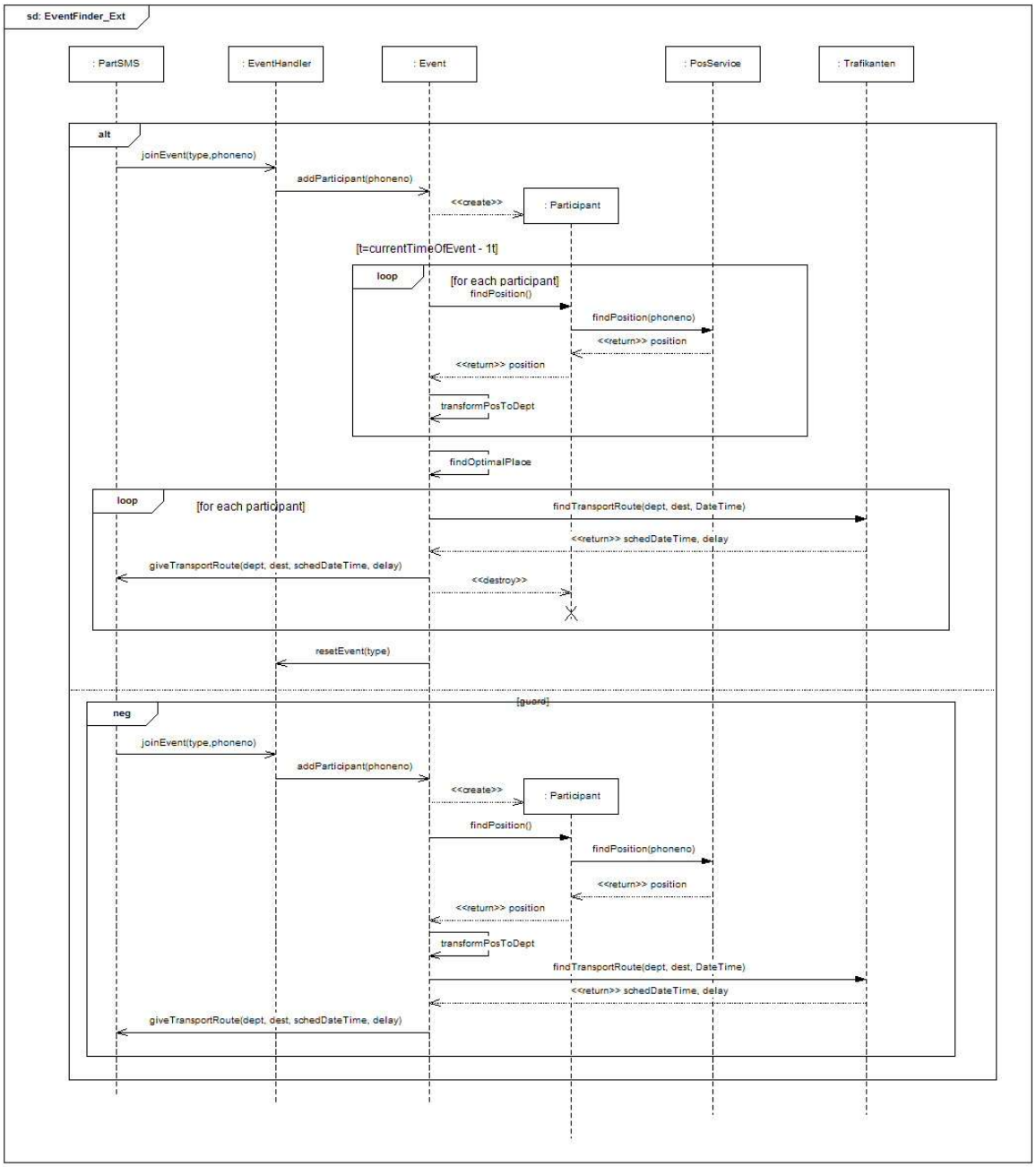

Figure 4 SD: EventFinder_Ext.

5.3. SD: EventFinder_Err

The sequence diagram EventFinder_Err describes how the system reacts if a participant tries to join a nonexisting event. In that situation the system returns an error message, and the participant will have to start all over again to join another event. If the event is found, the system will add the participant to the event, find position information and give the participant the transport route to get to the event.

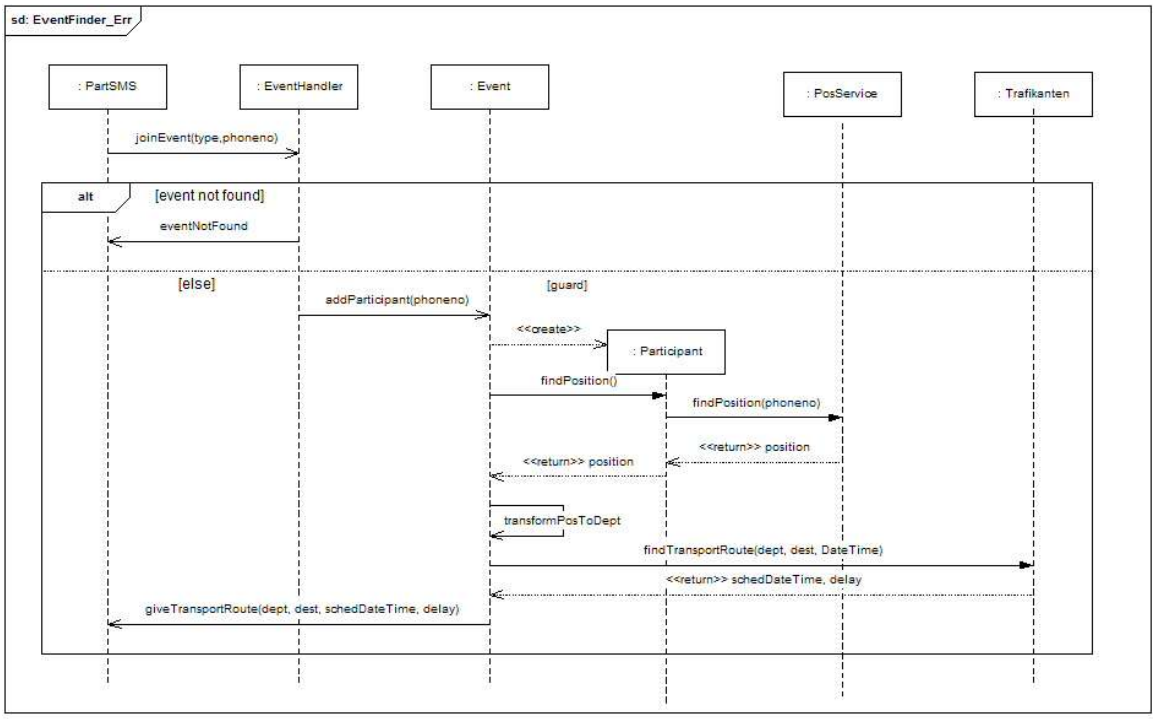

Figure 5 SD: EventFinder_Err.

6. The proof of refinement

We have tree different specifications--which we call S1, S2, S3--specifying the Multiple Blind Date System (MBDS). S1 specifies the abstract version of MBDS, where place is given. S1 consists of one diagram EventFinder_Abs. S2 has one diagram EventFinder_Ext and is an extended version of MBDS, which finds an optimal meeting place and specifies rules for the time trigger. S3 consists of diagram EventFinder_Err and stipulates error handling for message joinEvent(), where a participant tries to register for a nonexistent event.

S1, S2 and S3 do not contain xalt, therefore we have only one interaction obligation in each of the specifications.

6.1. EventFinder_Ext is refinement of EventFinder_Abs

The semantics [[S1]] for S1 should be in the form of [[S1]] = {(p1, n1)}, where
p1 = pos(EventFinder_Abs) and
n1 = neg(EventFinder_Abs)

p1 contains only one positive trace, t1, which is the trace where a participant joins an even and the system finds and returns a transport route to the event. n1 is an empty set.

Therefore the semantics [[S1]] for S1 will be in the form:
[[S1]] = {({t1}, Ø)}

The semantics [[S2]] for S2 should be in the form of [[S2]] = {(p2, n2)}, where
p2 = pos(EventFinder_Ext) and
n2 = neg(EventFinder_Ext)

p2 contains one positive trace, t2. Trace t2 is the *first* alternative which goes through a detailed version of S1. The trace remains positive if the time interval requirement is met. n2 contains two negative traces. The first negative trace, t1, is the trace for the *last* alternative in S2, which was a positive trace in S1. The second negative trace, t3, is the *first* alternative in S2 when the interval requirement is not met.

Therefore the semantics [[S2]] for S2 will be in the form:
[[S2]] = {({t2}, {t1, t3})}

According to the refinement definition of interaction obligations, we have that
$(p1, n1) \rightsquigarrow (p2, n2)$ iff

1. $n1 \subseteq n2$ and
2. $p1 \subseteq p2 \cup n2$

n1 is an empty set and therefore it is a subset of n2, thus the first requirement is met.

The trace in p1 is found in n2 hence the second requirement is met.

Therefore we have shown that (p1, n1) $\rightsquigarrow$ (p2, n2).
According to the definition of the property refinement, S2 is a refinement of S1, written S1 $\rightsquigarrow$ S2, iff

$\forall\ o \in [[S1]]: \exists\ o' \in [[S2]] :\ o \rightsquigarrow o'$

As mentioned earlier S1 and S2 have only one interaction obligation each. We have shown that (p1, n1) $\rightsquigarrow$ (p2, n2), hence S2 is a refinement of S1.

S2 is a supplement of S1 reached by transferring inconclusive traces in S1 to the sets of both positive and negative traces in S2. Narrowing has been done by transferring a positive trace in S1 to the set of negative traces in S2.

6.2. EventFinder_Err is refinement of EventFinder_Abs

The semantics [[S3]] for S3 should be in the form of [[S3]] = {(p3, n3)}, where
p3 = pos(EventFinder_Err) and
n3 = neg(EventFinderErr)

p3 contains two positive traces, t1 and t4. Trace t1 is the *last* alternative which is identical to the positive trace in S1. The second trace, t4, is the *first* alternative, which occurs if event is not found. n3 is an empty set.

Therefore the semantics [[S3]] for S3 will be in the form:
[[S3]] = {({t1, t4}, Ø)}

According to the refinement definition of interaction obligations, we have that
(p1, n1) $\rightsquigarrow$ (p3, n3) iff

1. n1 $\subseteq$ n3 and
2. p1 $\subseteq$ p3 $\cup$ n3

n1 is an empty set and therefore it is equal to n3, thus the first requirement is met.

The trace in p1 is found in p3, thus the second requirement is met.

Therefore we have shown that (p1, n1) $\rightsquigarrow$ (p3, n3).
According to the definition of the property refinement, S3 is a refinement of S1, written S1 $\rightsquigarrow$ S3, iff

$\forall\ o \in [[S1]]: \exists\ o' \in [[S3]] :\ o \rightsquigarrow o'$

As mentioned earlier, S1 and S3 have only one interaction obligation each. We have shown that (p1, n1) $\rightsquigarrow$ (p3, n3) hence S3 is a refinement of S1.

S3 is a supplement of S1 reached by transferring inconclusive traces in S1 to the set of positive traces in S3.