UNIVERSITY OF OSLO
Department of Informatics

INF5150 Group No. 5 Multiple Blind Date

Rune Froeysa, runefr,
Ina Flesvik, inahe,
Vikash Katta, vikashk,
Kristoffer Stav, krsta

14th October 2005

Contents

1 Introduction

This document describes the specifications for the Multiple Blind Date system, using UML 2.0 diagrams.

Multiple Blind Date system is a service that organizes events for customers who participate. The customers subscribes for events via SMS. A reminder message is sent back to the customers x minutes prior to the event with a suggested public transport schedule.

Section4 describes a general approach for the Multiple Blind Date system, which includes a use case diagram, class diagram, composite structure diagram and multiple sequence diagrams. Section 5 discuss some alternative solutions as refinements of the general one, with sequence diagrams and proofs.

We have used IBM Rational Software Modeler for designing diagrams. It lacks some of the functionality we would like do include. An example is passing messages from one sequence diagram to another. There is no way to draw a message from the diagram edge to a lifeline (that we know of). A workaround for this problem was to make a lifeline that represent the diagram edge. Not very elegant, but it does the work.

2 Assumptions

In our first approach of designing the Multiple Blind Date system we assume that each event is commenced on a fixed location. Under Section 5 we have presented an alternative solution where the meeting place is decided based on the client's positions.

The customers communicate with the system via SMS, and the phone-number is the customer identification.

The data format on the position is not discussed in this document. Whether the positions are represented with x and y coordinates, or represented by names stored as strings, is not defined here.

Multiple Blind Date needs more features than the ones described in this document before going into production, including:

- A back-end interface for system maintenance, i.e adding and removing events.
- An information provider is needed, i.e. a web-site which lists up all available events the customers can sign up for, including a description of how to sign up. The site could fetch all entries listed up in the table Events.

3 The main system components and its context

This section gives a brief description of the components in the Multiple Blind Date system.

Client is the interface towards the customers which signs up for events via SMS sent to a given phone-number. The client listens for messages from customers, and sends messages back to the customers.

BlindDateSystem (BDS) is the components which manages and stores information about events and clients signed up for events. Time, position and participants are kept for each event.

Trafikanten is an external service. It is used to find a public transportation route, time schedule and delays for the customers. From-position (client position) and to-position in addition to the time is required to retrieve this information.

PositioningSystem is an external service which finds the client's position. The phone-number is sent to the PositioningSystem and the position is returned.

Timer triggers the BDS to send out event notifications (with a suggestion for a public transportation schedule) x minutes prior to the event.

Event is an entity that represent an event. Each event has a time, location, and a list of clients subscribed for that event.

4 A general Multiple Blind Date system

4.1 Description

Our first approach describes a general Multiple Blind Date service. As mentioned in the introduction, the location for the events is fixed. How the clients interact with the system, and how the system interact with the clients are illustrated in the use case diagram in Figure 1 in Section 4.2.

An instance of BDS is the main component. It Ontarians instances of a Controller and a list of events. It also has references to the external components Client, Trafikanten, the Timer and the PositioningSystem.

4.2 Use case diagram

The use case diagram in Figure 1 shows the clients and how they interact with the system. It also shows what tasks the BDS system performs.

There are mainly two use cases; subscribe event is the scenario where a client subscribes for an event. An acknowledgement is sent from BDS to the client in return. The second use case is the scenario where BDS sends the public transport schedule to the client.

4.3 Composite structure diagram

Composite structure diagrams shows how components collaborate with each other over communication links. Figure 2 shows a composite structure diagram for the

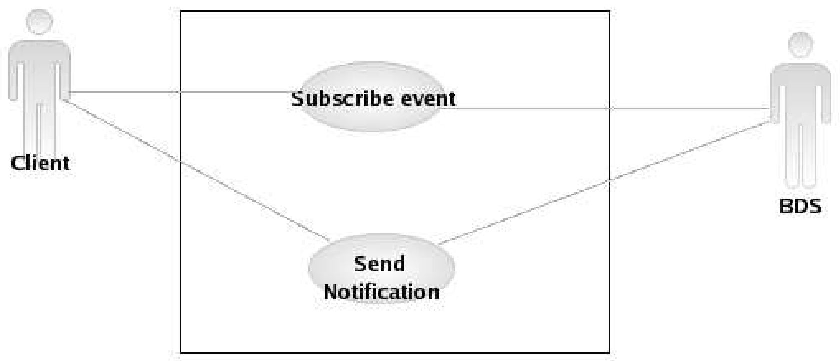


Figure 1: Use case diagram

Multiple Blind Date system. The components Client, Trafikanten, PositioningSystem and Timer are external components.

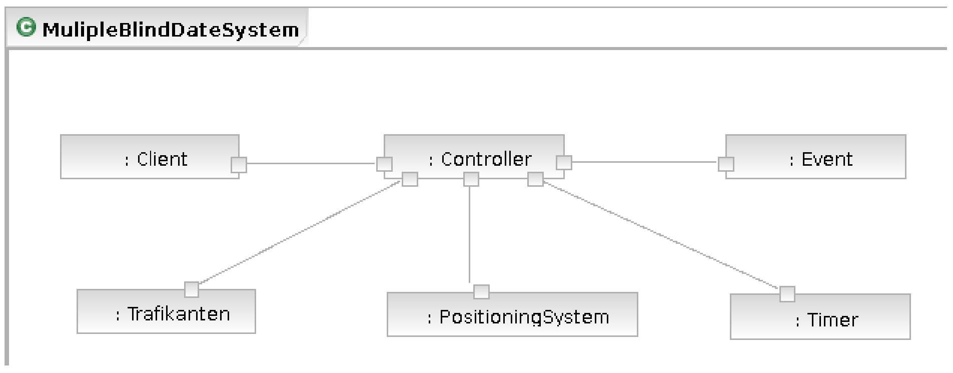


Figure 2: Composite structure diagram

4.4 Class diagram

Figure 3 shows the class diagram for the Multiple Blind Date System. Read about these classes in Section 3. The controller is the component which takes care of decision making and communication with the various components.

4.5 Sequence diagrams

The sequence diagrams illustrate the chronological sequence of messages and interactions between the components in the system. The first diagram shows an overview over the main sequence diagrams, see Figure 4.

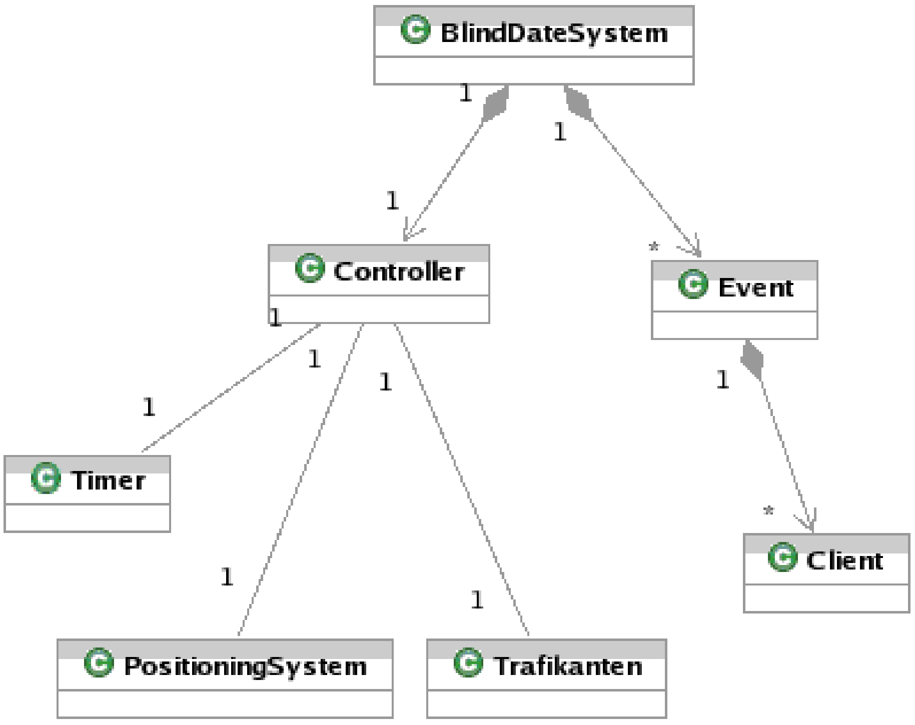


Figure 3: Use case diagram

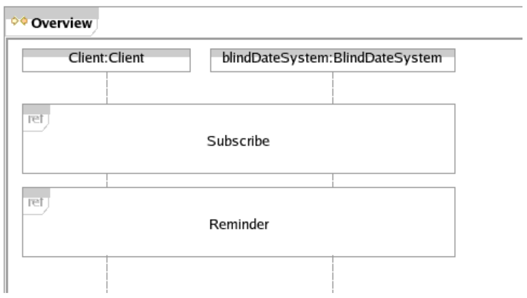


Figure 4: Overview

The sequence diagrams Subscribe and Reminder, which is referenced in Figure 4 is illustrated in Figure 5 and 6, respectively.

The subscribe diagram shows the interaction between the client and BDS, when the client subscribes for an event. See Figure 5.

The reminder diagram shows the interaction betweem the various components of the system, including Trafikanten and the PositioningSystem in order to send a transport schedule to all the clients subsribed for an event. The reminder is triggered by a Timer. See Figur 6.

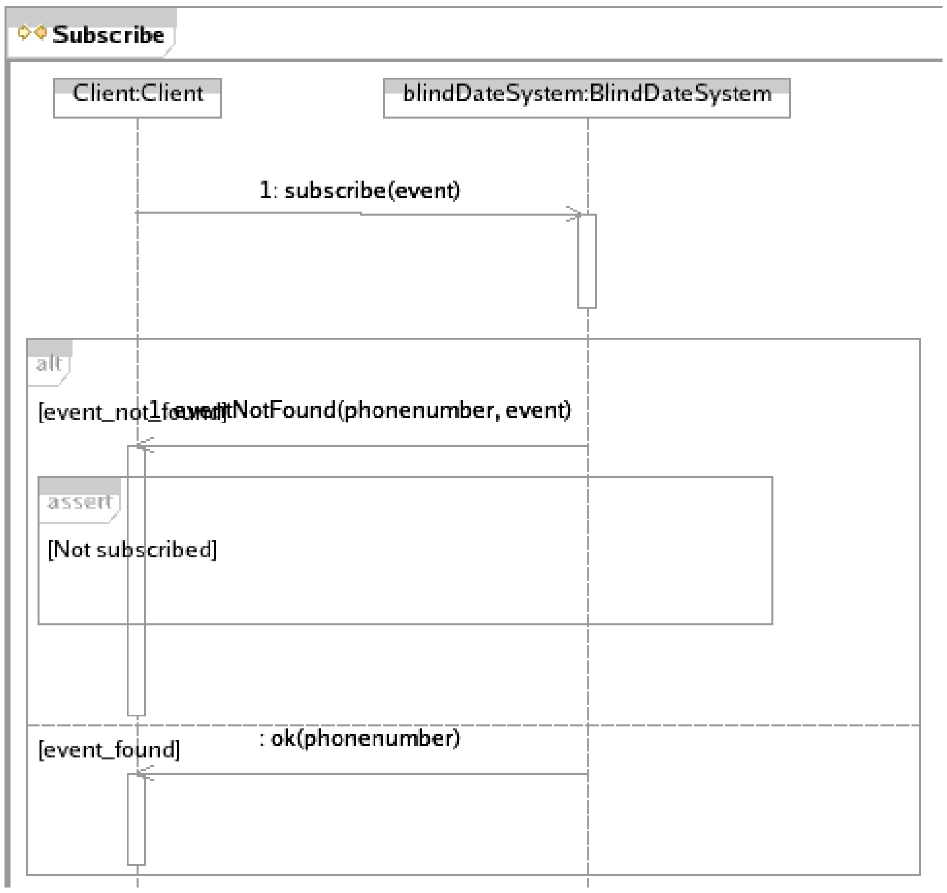


Figure 5: Subscribe

Decomposition of the lifeline BlindDatesystem in Figure 4 regarding Subscribe and Reminder is illustrated in the Figures 7 and 8, respectively[1]

Figure 9 shows how the Controller communicates with the PositioningSystem and Trafikanten in order to come up with a suggestion for a public transportation schedule. Client position is retrieved from the PositioningSystem, which we can assume find the position of the client's mobile phone. When this information is retrieved, client position, event location and the time for the event is sent to Trafikanten. A description is returned, which is sent back to the client. As we can see of the alt fragment in Figure 9, Trafikanten might not come up with a route suggestion because of some reason, then this is expressed with ‘schedule not found‘.

[1]There is no way to make a **ref** in the lifeline head (as far as we know).

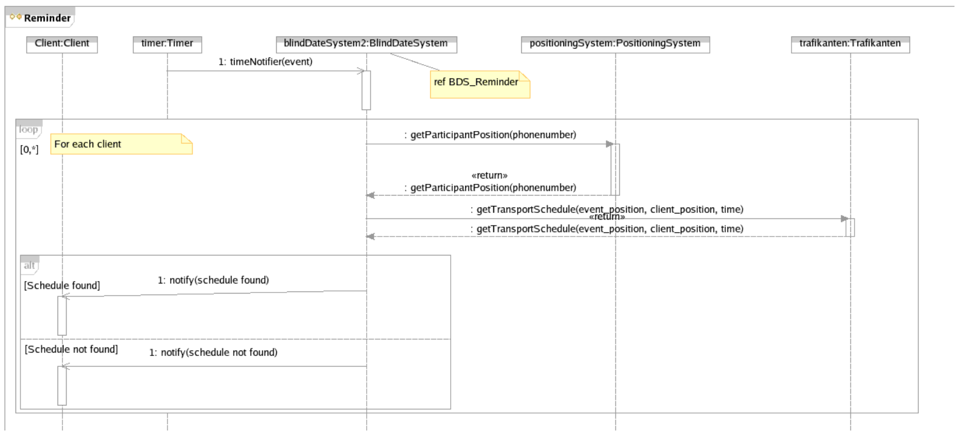

Reminder
Client:Client
timer:Timer
blindDateSystem2:BlindDateSystem
positioningSystem:PositioningSystem
trafikanten:Trafikanten
1: timeNotifier(event)
ref BDS_Reminder
loop
[0,*]
For each client
: getParticipantPosition(phonenumber)
«return»
: getParticipantPosition(phonenumber)
: getTransportSchedule(event_position, client_position, time)
«return»
: getTransportSchedule(event_position, client_position, time)
alt
[Schedule found]
1: notify(schedule found)
[Schedule not found]
1: notify(schedule not found)

Figure 6: Reminder

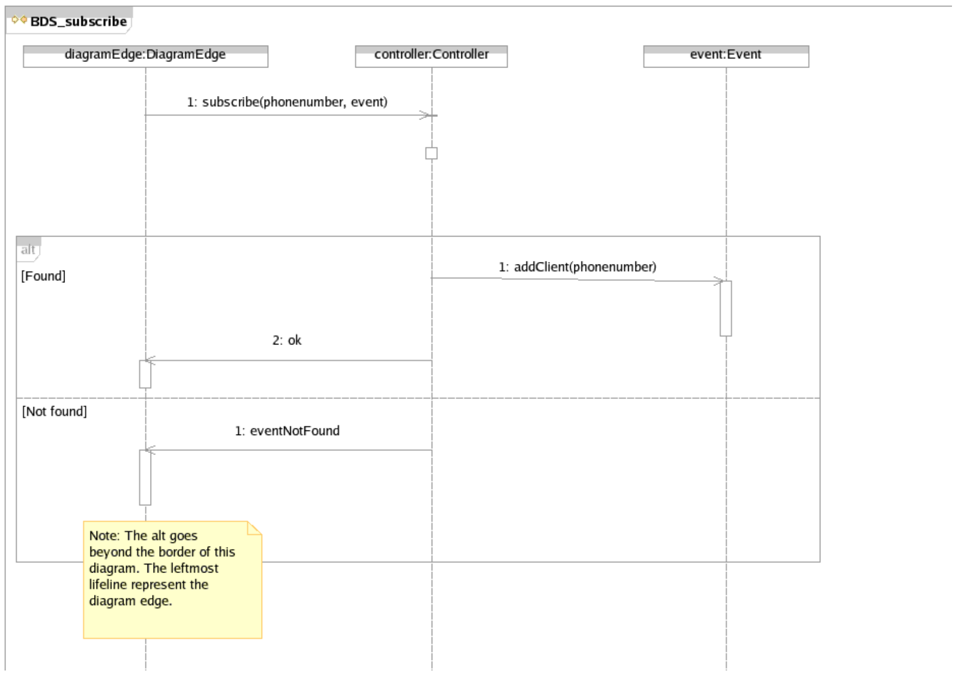

BDS_subscribe
diagramEdge:DiagramEdge
controller:Controller
event:Event
1: subscribe(phonenumber, event)
alt
[Found]
1: addClient(phonenumber)
2: ok
[Not found]
1: eventNotFound
Note: The alt goes beyond the border of this diagram. The leftmost lifeline represent the diagram edge.

Figure 7: Decomposition of MultipleBlindDateSystem: subscribe

5 Alternative solutions

We have made three alternative solutions for the Multiple Blind Date system. The changes are described in the following subsections. We have included diagrams here that illustrate changes compared to the original design. In Section 5.1 we see

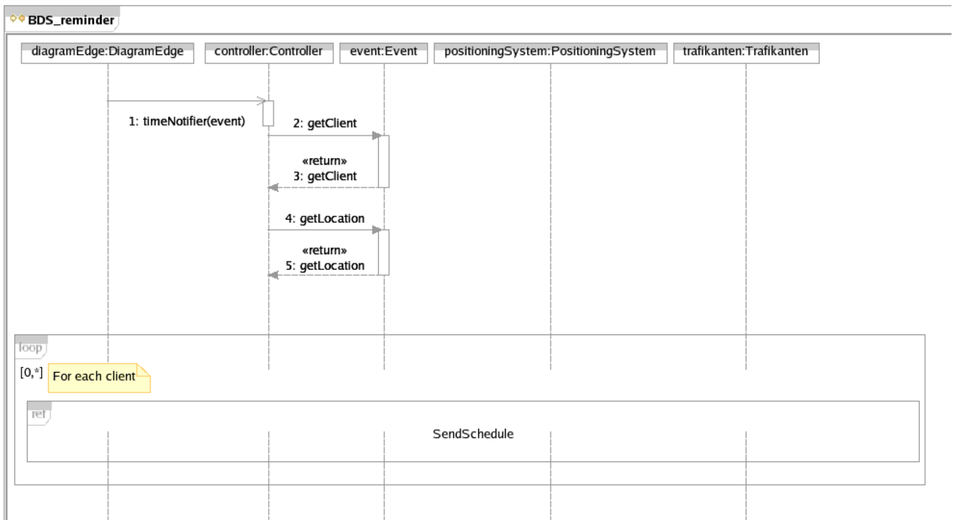


Figure 8: Decomposition of MultipleBlindDateSystem: reminder

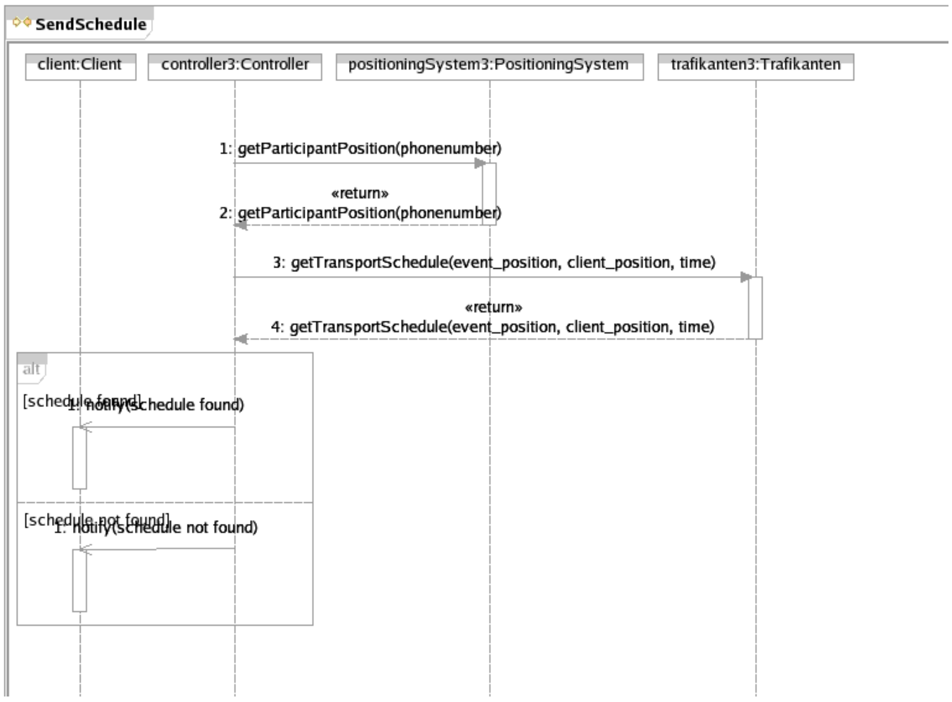

SendSchedule
client:Client
controller3:Controller
positioningSystem3:PositioningSystem
trafikanten3:Trafikanten
1: getParticipantPosition(phonenumber)
«return»
2: getParticipantPosition(phonenumber)
3: getTransportSchedule(event_position, client_position, time)
«return»
4: getTransportSchedule(event_position, client_position, time)
alt
[schedule found]
1: notify(schedule found)
[schedule not found]
1: notify(schedule not found)

Figure 9: Send public transportation schedule

why these alternative solutions are refinement of the original service description.

5.1 Refinement and proofs

First, we will very briefly present the requirements for refinement stated in STAIRS. Refinement involves making modifications to a system design through three approaches:

Supplementing : define more negative or positive traces in the system. Or said with other words: make inconclusive traces positive or negative. In practice you add features to the system, or define features that are not allowed in the system

Narrowing : involves reducing the number of positive traces, by make a subset of the positive traces negative. In practice you are reducing the number of "features" or positive runs in the system, by putting some constraints to it.

Detailing involves giving a more detailed description, without changing the existing behavior considerably.

We will present give alternative solutions to the system. These alternative solutions have made changes to parts of the system.

5.2 Immediate event

Our first approach of modify the original design is to include a new feature as follows: if the event is commencing within 30 minutes from the subscription, no reminder/notification is given. Instead the transport schedule is sent right away.

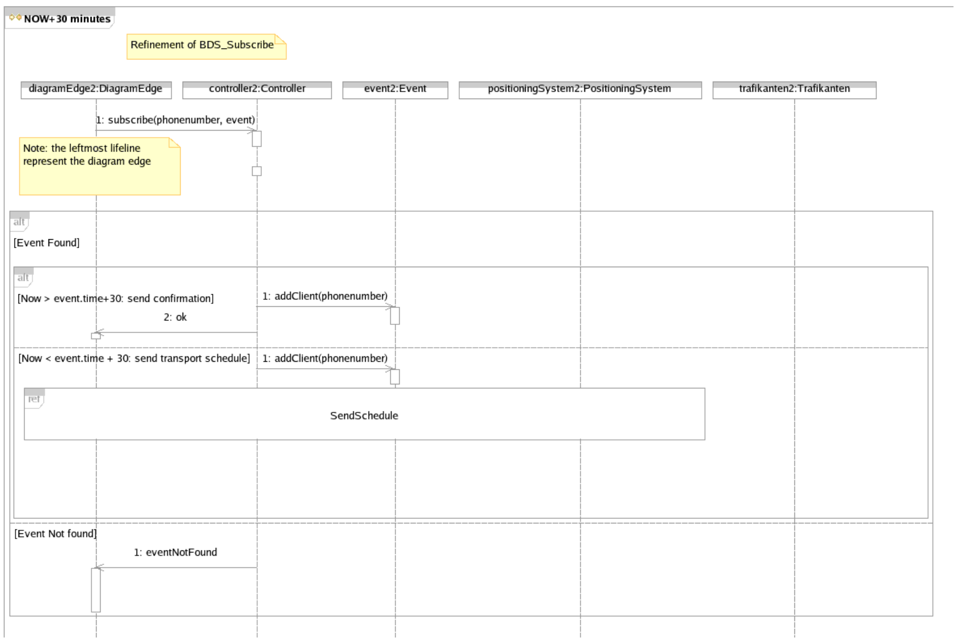
The modification is a refinement of BDS_subscribe presented in Figure 7. A modified sequence diagram for event subscribing is given in Figure 10. As we can see, a check is performed within the innermost alt fragment: if the event is occurring within 30 minutes, the "send schedule routine" is called, if not, an ordinary confirmation (stated with "OK" in the diagram) is returned to the client, and the transport schedule is sent later.

This is a refinement of the original diagram, because we have introduced some new positive traces. This is supplementing. The positive traces in the original solution is a subset of the traces in this solution. The same applies to the other alternatives in the folling subsections.

5.3 External services are down

In our original system description we have not taken into account that the external components Trafikanten and the PositioningSystem can be unavailable for some reason. A modification of SendSchedule is illustrated in Figure 11.

This is a refinement of the original diagram. The inconclusive traces (Service is down) are made negative, and we have not introduced additional positive traces. According to refinement defined by STAIRS, this is supplementing, which makes the sequence diagram presented in Figure 11 a refinement of the original diagram.



NOW+30 minutes
Refinement of BDS_Subscribe
diagramEdge2:DiagramEdge
controller2:Controller
event2:Event
positioningSystem2:PositioningSystem
trafikanten2:Trafikanten
1: subscribe(phonenumber, event)
Note: the leftmost lifeline represent the diagram edge
alt
[Event Found]
alt
[Now > event.time+30: send confirmation]
1: addClient(phonenumber)
2: ok
[Now < event.time + 30: send transport schedule]
1: addClient(phonenumber)
ref
SendSchedule
[Event Not found]
1: eventNotFound

Figure 10: Refinement of subscribe

5.4 Too few subscribers

A third alternative solution to the original system design is to cancel an event if too few customers has subscribed., see Figure 12. If less than two (just a arbitrary number), the event is canceled. This alternative solution involves adding new traces to the reminder sequence diagram, hence this is supplementing.

5.5 The most optimal meeting place for the participants

The EventManager will check the position for every participant and find a location which is most optimal for each participant, as we can see in Figure 13. We assume that the controller has sufficient logic to calculate the optimal meeting place by using a list of the clients position. The Controller will then ask for travel info from Trafikanten to this optimal place, and send travel info to each participant.

As in the previous refinements, we have introduced new positive traces. This is supplementing. No negative traces are defined.

5.6 Unsubscribe

The clients should have the ability to cancel the participation for events. This is illustrated in Figure 14 and 15. This involves adding new features, hence it is supplementing according to STAIRS.

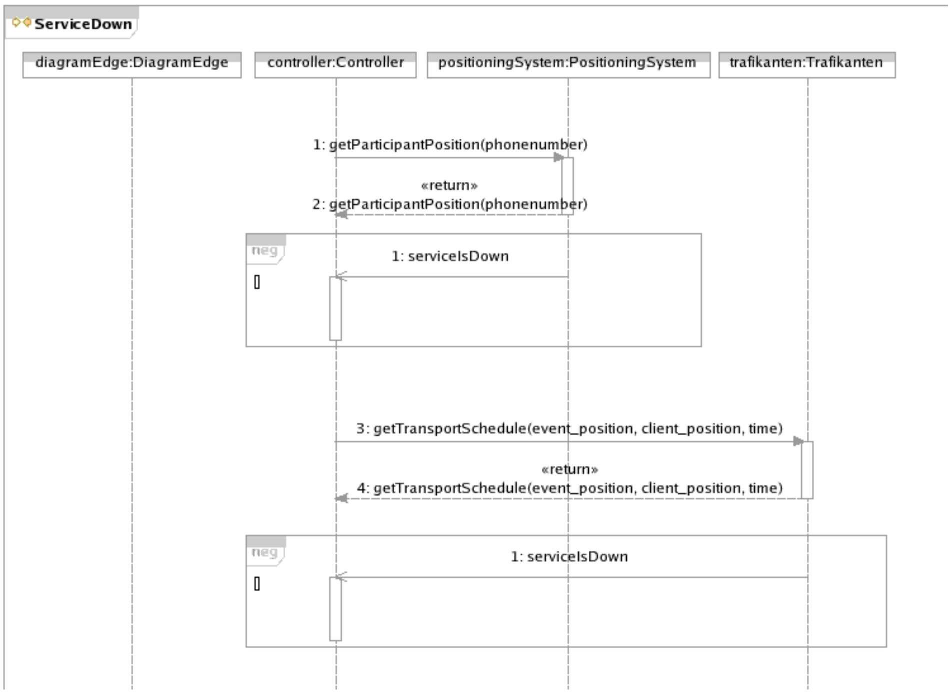

ServiceDown
diagramEdge:DiagramEdge
controller:Controller
positioningSystem:PositioningSystem
trafikanten:Trafikanten
1: getParticipantPosition(phonenumber)
«return»
2: getParticipantPosition(phonenumber)
neg
1: serviceIsDown
3: getTransportSchedule(event_position, client_position, time)
«return»
4: getTransportSchedule(event_position, client_position, time)
neg
1: serviceIsDown

Figure 11: Refinement of SendSchedule

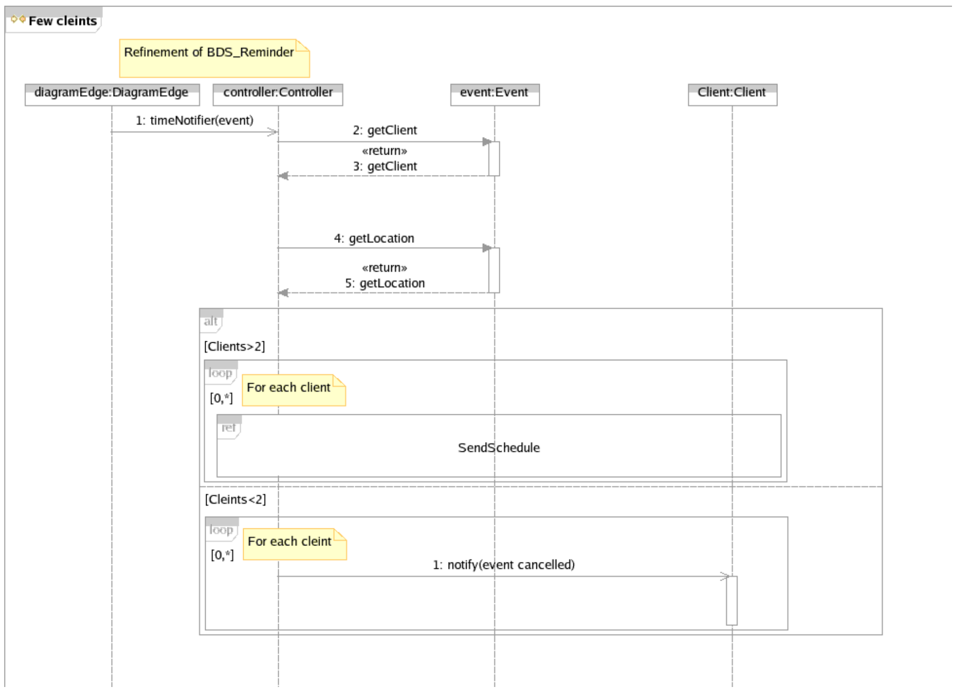


Figure 12: Too few subscribers

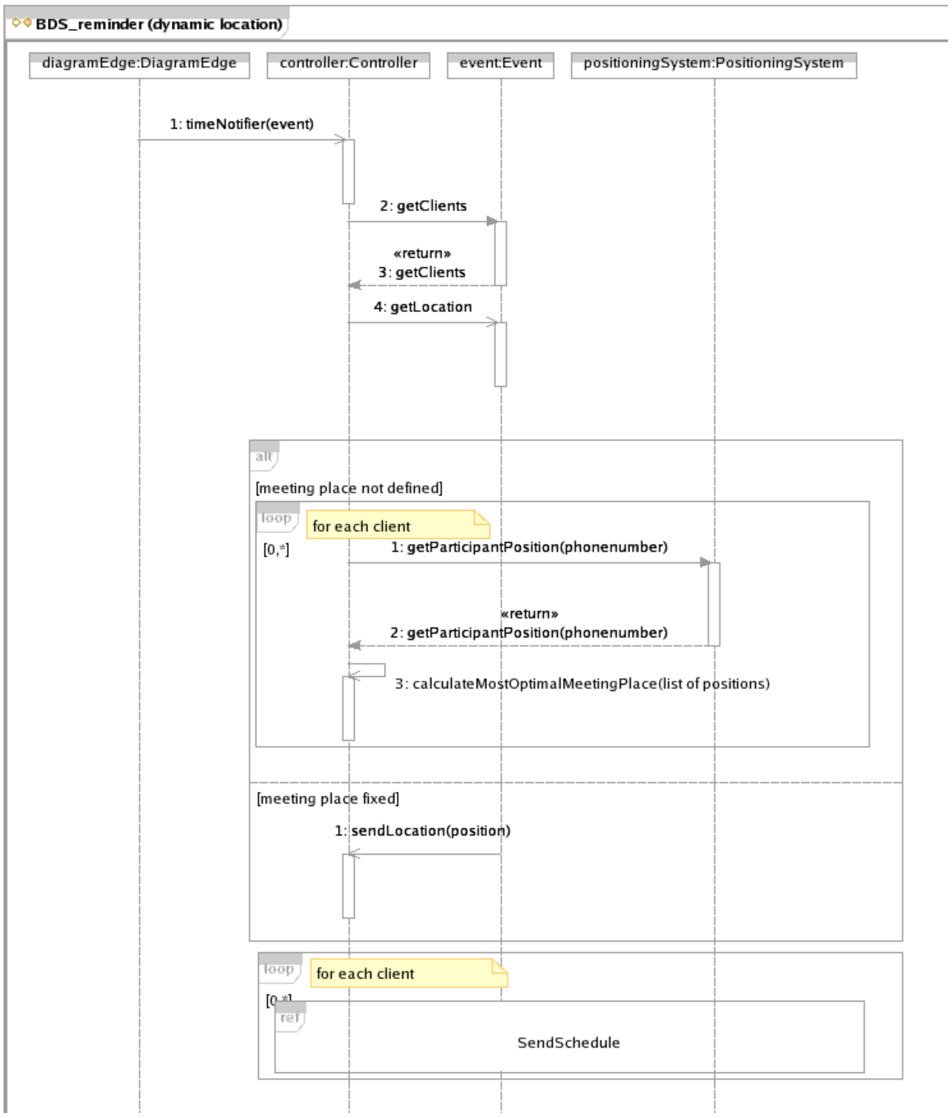

BDS_reminder (dynamic location)
diagramEdge:DiagramEdge
controller:Controller
event:Event
positioningSystem:PositioningSystem
1: timeNotifier(event)
2: getClients
«return»
3: getClients
4: getLocation
alt
[meeting place not defined]
loop
for each client
[0,*]
1: getParticipantPosition(phonenumber)
«return»
2: getParticipantPosition(phonenumber)
3: calculateMostOptimalMeetingPlace(list of positions)
[meeting place fixed]
1: sendLocation(position)
loop
for each client
[0,*]
ref
SendSchedule

Figure 13: Optimal meeting place: refinement of BDS_reminder

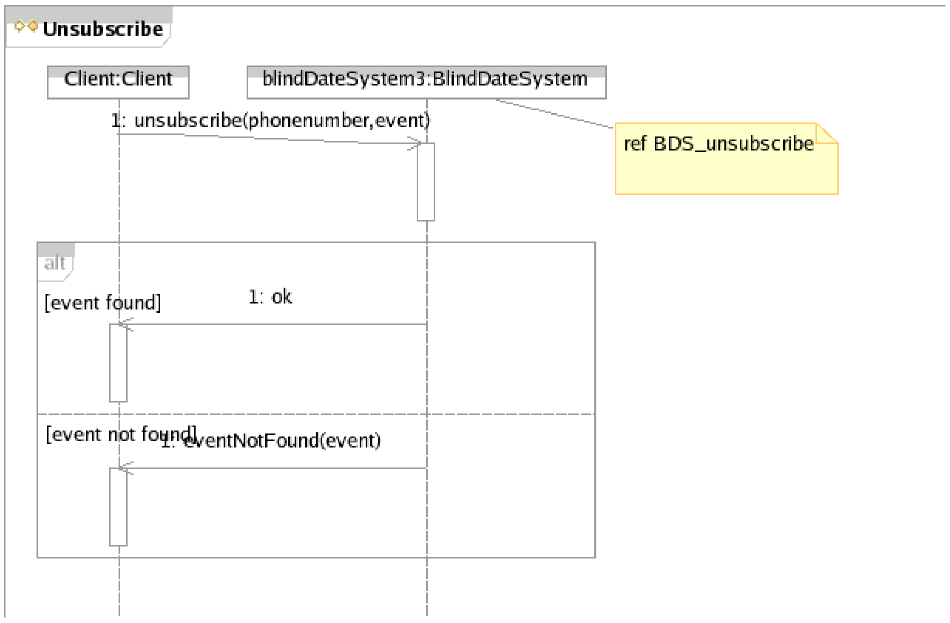


Figure 14: Refinement: new functionality added, subscribe

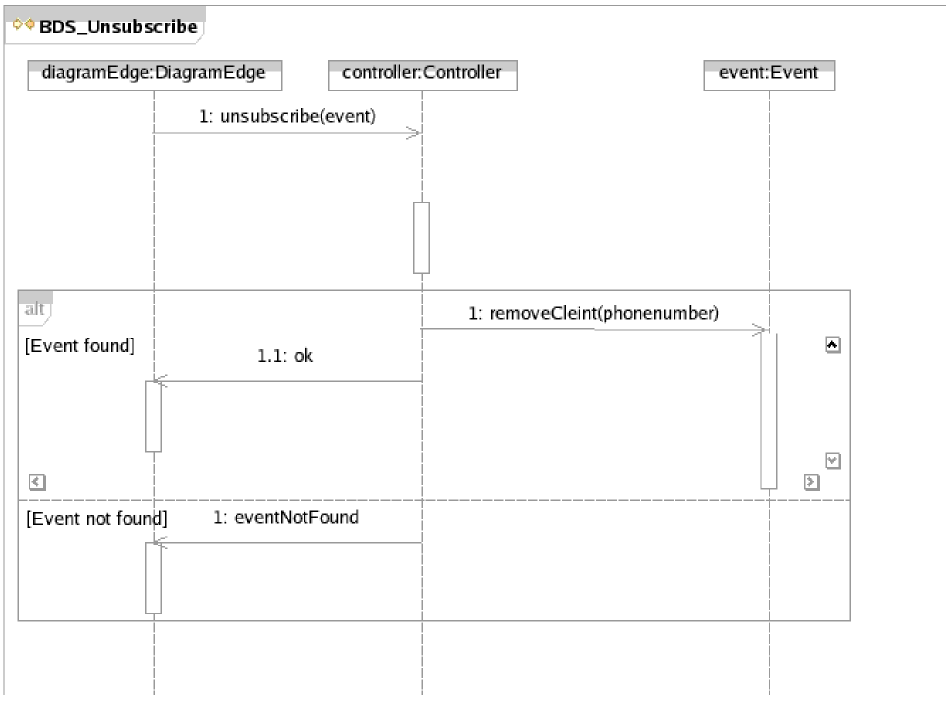


Figure 15: Subscribe