INF5150 Obligatory exercise Drop 1
Proposed solution
October 21, 2005

Introduction

We are going to make a system (called BlindDate) to meet a group of people. The customers (participants) register for events by the use of SMS messages. At appropriate time before an event, the system sends a message containing traveling information to each participant depending on her/his location.

Three different specifications are given:

- BlindDate1 is the most general variant. It offers functionality that enables the customer to register in the system, join a particular event and receive notification with travel information. All events happen at a fixed and known location, independently of the location of the customers.
- BlindDate2 is identical with BlindDate1, with the three exceptions: Firstly, an event may be created by initiative from a customer. Secondly, the system may refuse a customer if the maximal number of allowed participants for a certain event is exceeded. Thirdly, notification messages are always triggered 30 minutes before the start of the event.
- BlindDate3 is identical with BlindDate1, except that a new option has been added: The location of the customers at a given time (shortly before the event) may now determine where the event will take place.

Both BlindDate2 and BlindDate3 will be shown to be refinements of BlindDate1.

Assumptions

The following assumptions are made:

1. All customers are identified by their phone number.
2. In cases where the system needs to communicate with more than one customer to accomplish a certain task, two customers represent collection of all relevant customers.
3. Existing events contain information about the location name (i.e. a restaurant) and the name and ID of the nearest bus stop. An exception to this is BlindDate3, since in this specification the location of the event is not decided until shortly before the event.
4. Events are published independently of the system, and are known by all customers. An exception to this is events initiated by a customer in BlindDate2.
5. Positioning requests from the system are sent in similar way as ordinary SMS messages. Therefore also positioning messages are shown as going to/from the customer’s mobile.
6. There are a fixed number of locations where events may take place. These locations are stored in LocationSupplier, together with their closest bus stop.

Naming conventions

Representation of SMS messages in the sequence diagrams

In JavaFrame the SMS messages will have format *Sms(String: to, String: from, String: message).* In the sequence diagrams SMS messages are written on the format *Sms(phoneNo, keyword, arg_1, … ,arg_n),* where *phoneNo* represents the customer’s phone number. The number of arguments depends on the keyword. The message string in the JavaFrame format corresponds to the concatenation of the keyword followed by all the arguments, with spaces in between.

Here is an example: The message *Sms(phoneNo, join, eventType, time)* in a sequence diagram may be instantiated for example by *Sms("2034","99999999", "Join drink beer at 20.30 Saturday").* 99999999 is the phone number of the joining customer. The field "2034" is not included in the sequence diagram SMS messages, since it is the same for all messages.

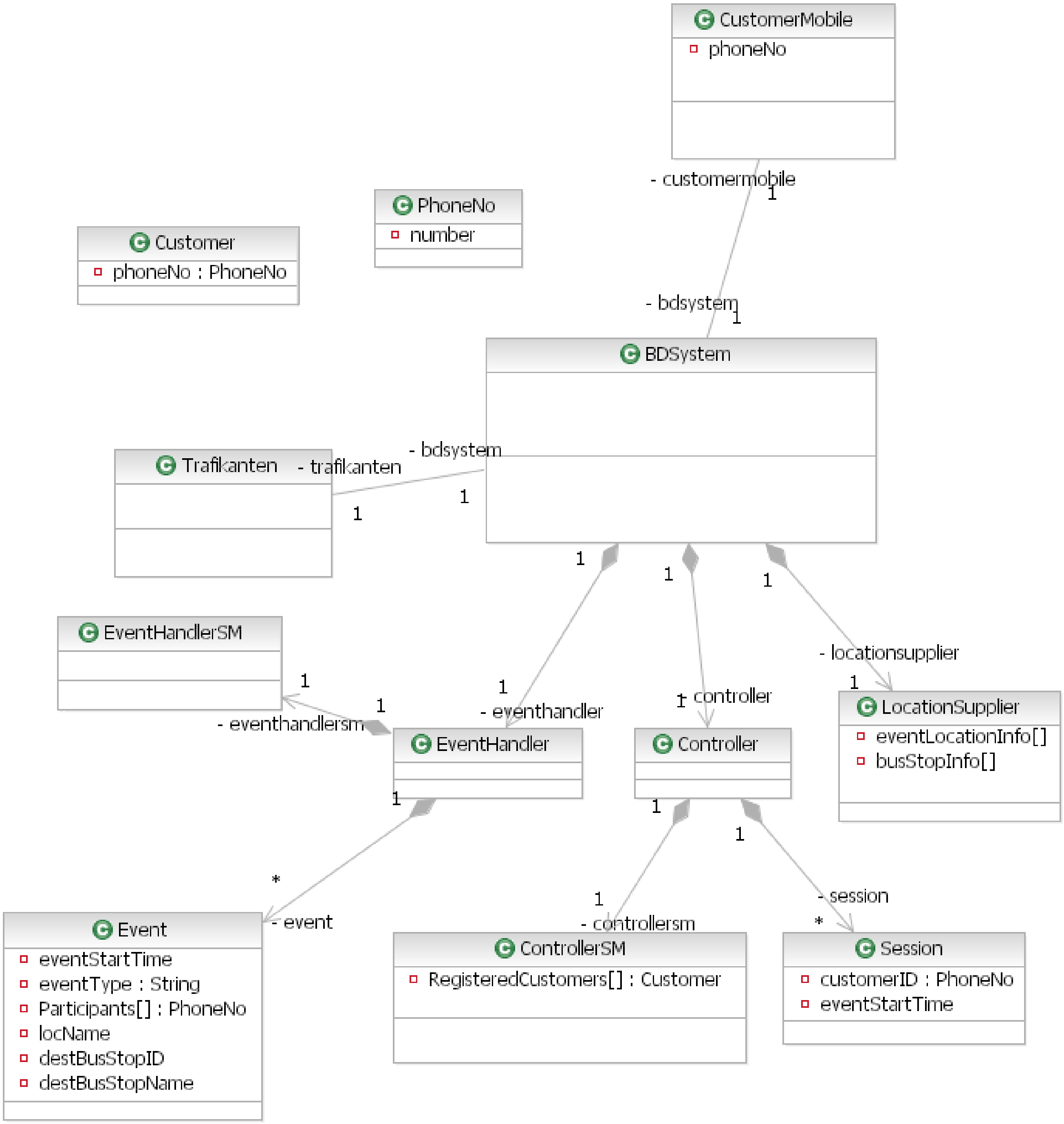
Message arguments

The following naming conventions apply to arguments of messages (this is an incomplete list):

- eventType: For example "drink beer", "play bridge" etc
- depBusStopName: The name for the bus stop the customer is advised to travel from, for example "Bjølsen"
- depBusStopID: A unique ID for the above bus stop name. Information about bus stop IDs is stored in LocationSupplier. The ID is used in DynRequest messages to Trafikanten.
- destBusStopName: The name for the bus stop the customer is advised to travel to.
- destBusStopID: A unique ID for the above bus stop name.
- locName: The name of the place where participants are supposed to meet, for example "Café Hansen"
- routes: Information returned from Trafikanten. See javadoc for communication with Trafikanten.

Class Diagram

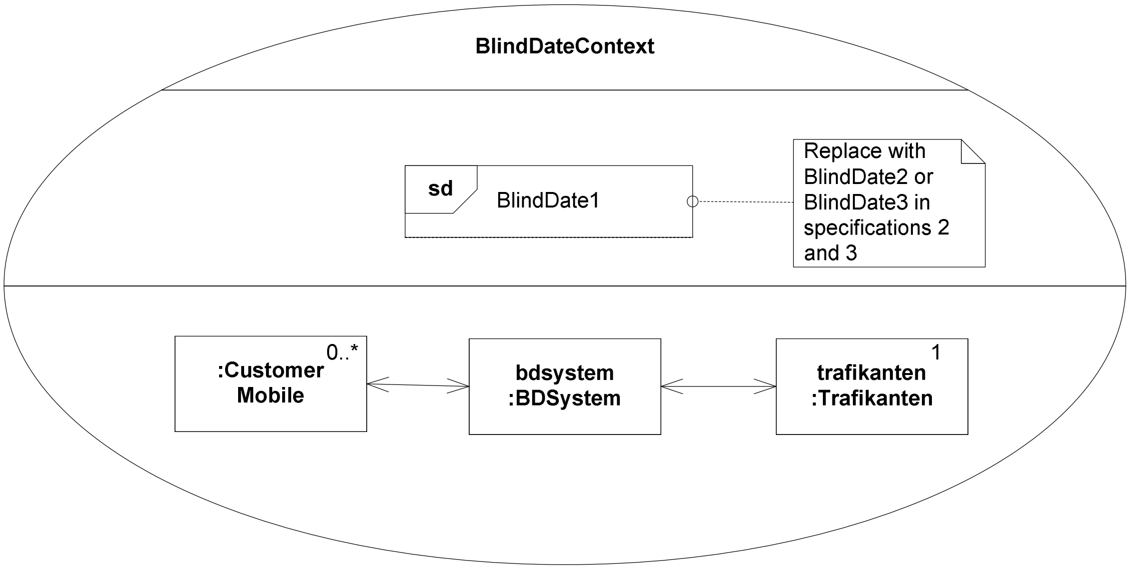
Note that not all attributes are necessarily shown.



BlindDate1

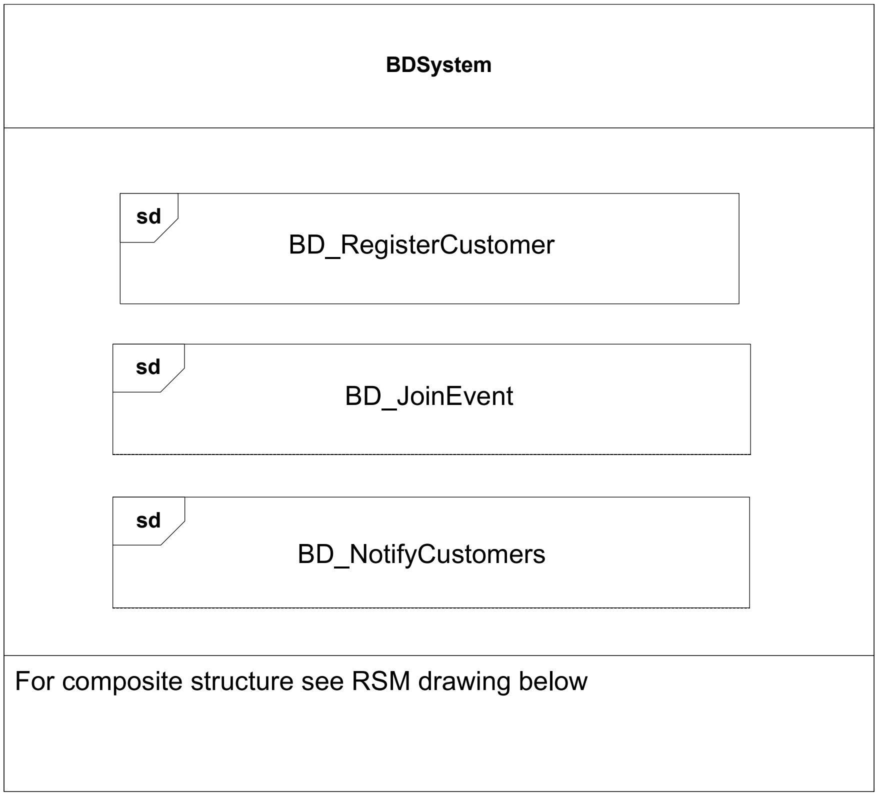
This section describes the diagrams relevant for BlindDate1.

BlindDateContext

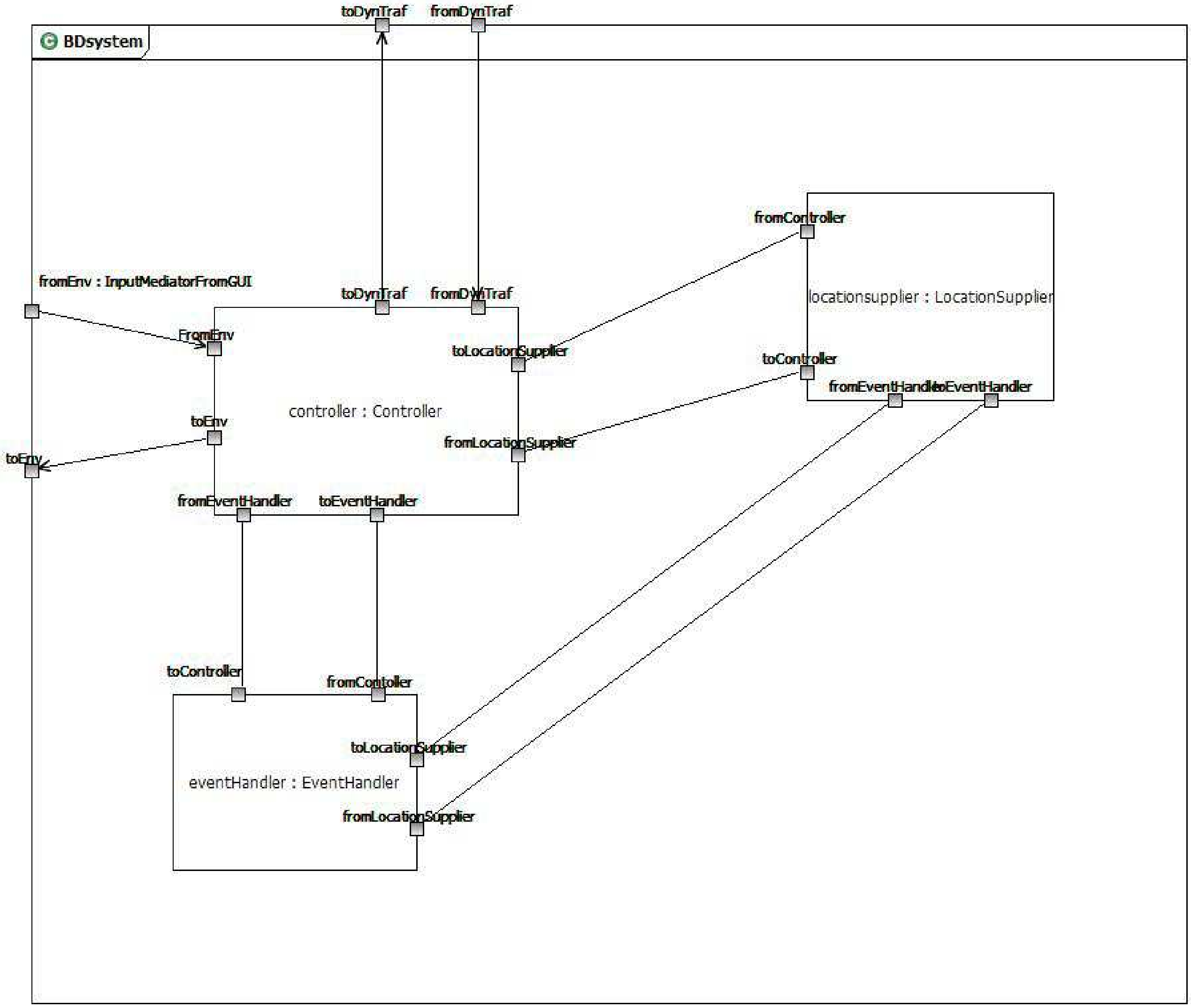


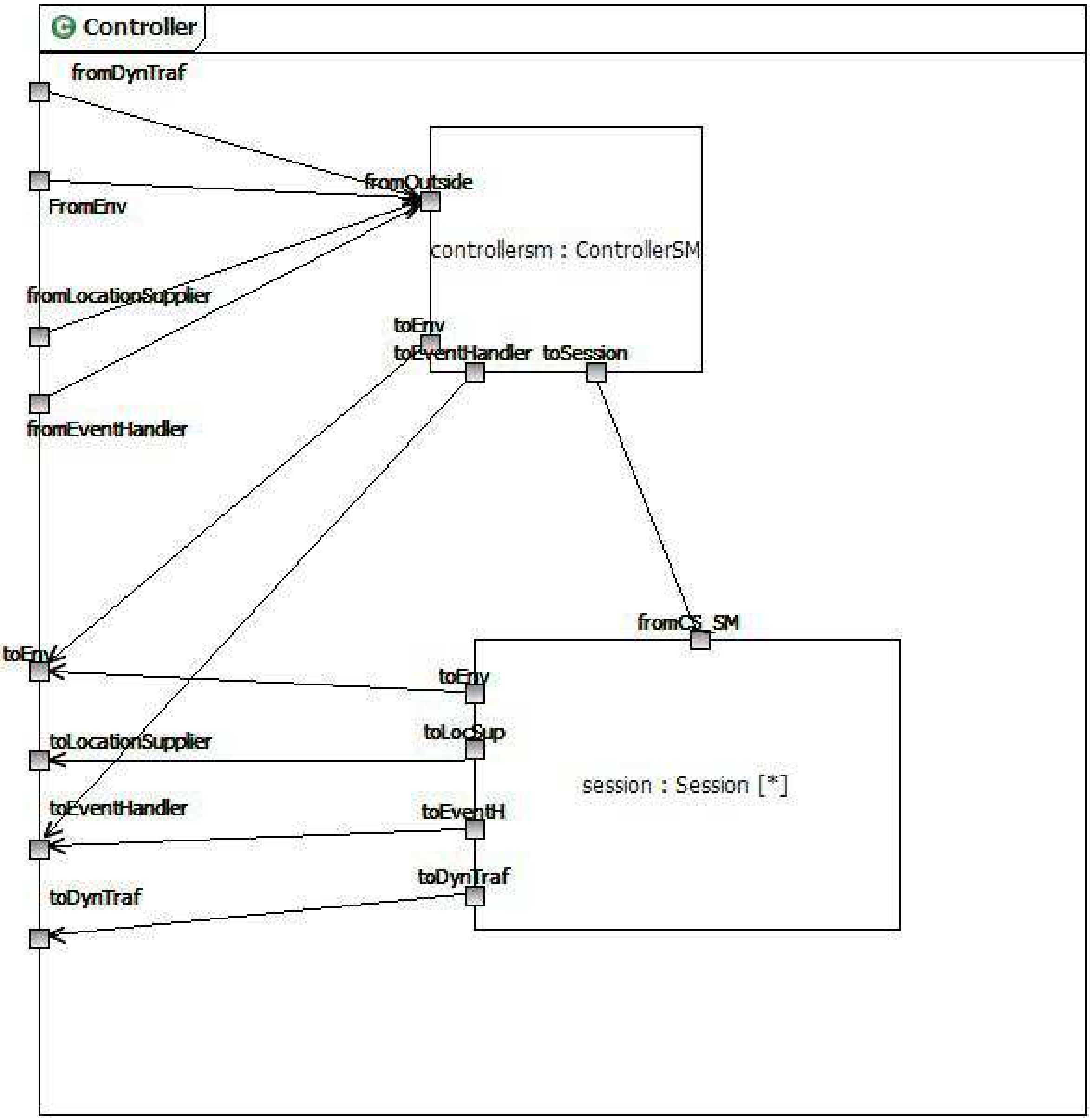
The diagram BlindDateContext is similar to all the specifications, except that sequence diagram BlindDate1 is replaced by BlindDate2 and BlindDate3.

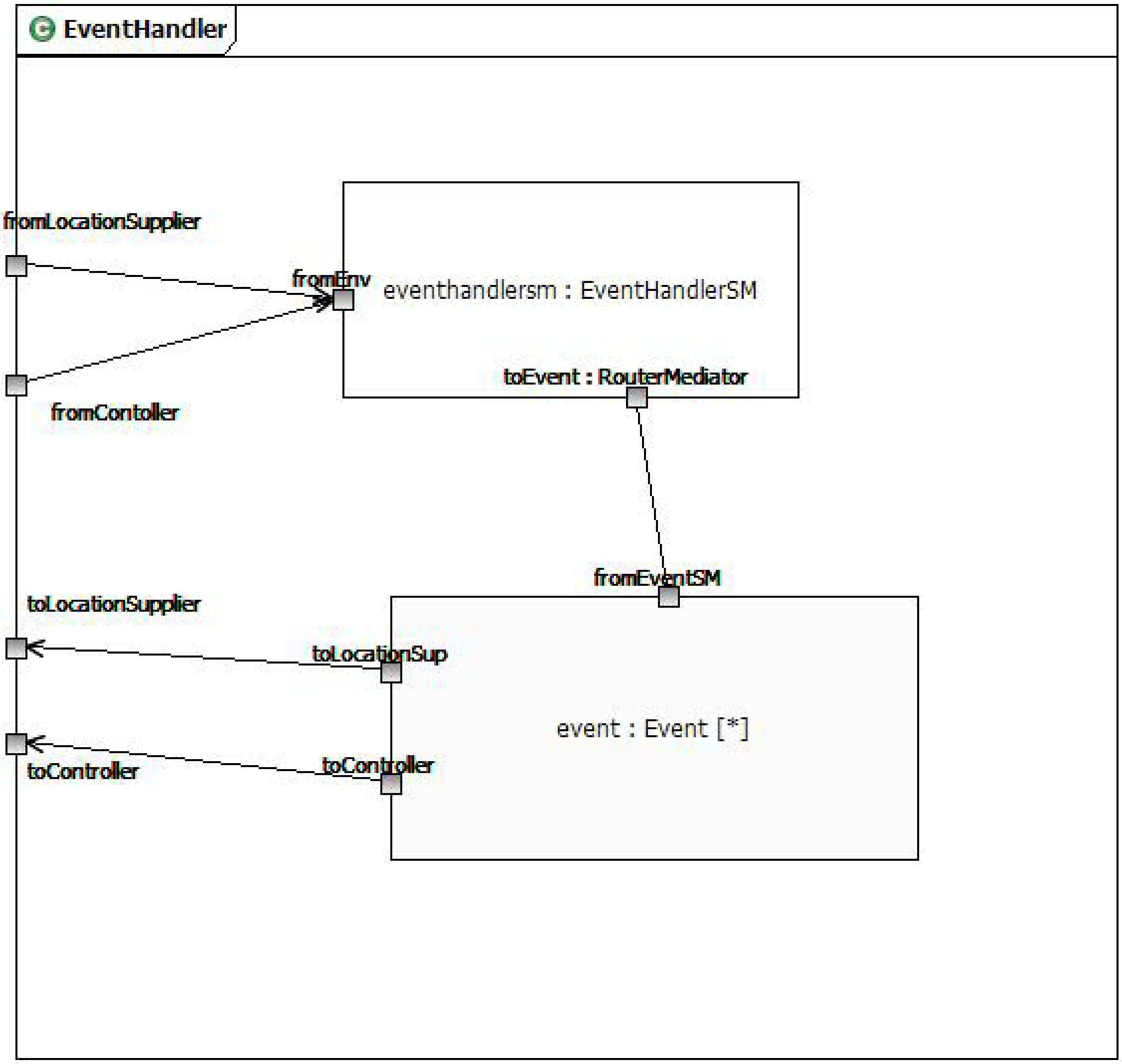
BDSystem structure



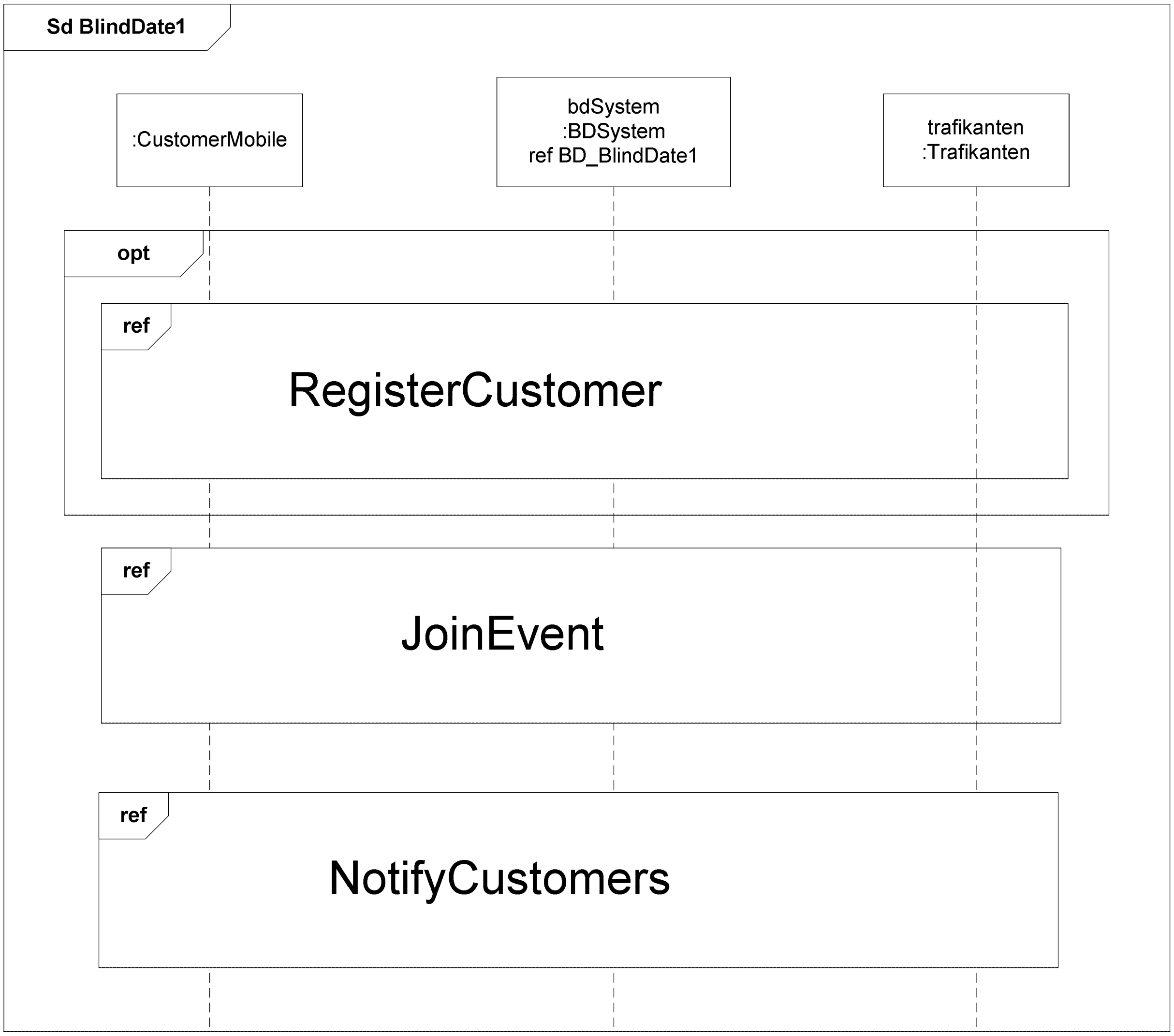
For composite structure see RSM drawing below


toDynTraf
fromDynTraf
BDsystem
fromController
locationsupplier : LocationSupplier
toController
fromEventHandler
toEventHandler
fromEnv : InputMediatorFromGUI
toDynTraf
fromDynTraf
FromEnv
toLocationSupplier
controller : Controller
toEnv
toEnv
fromLocationSupplier
fromEventHandler
toEventHandler
toController
fromContoller
toLocationSupplier
eventHandler : EventHandler
fromLocationSupplier

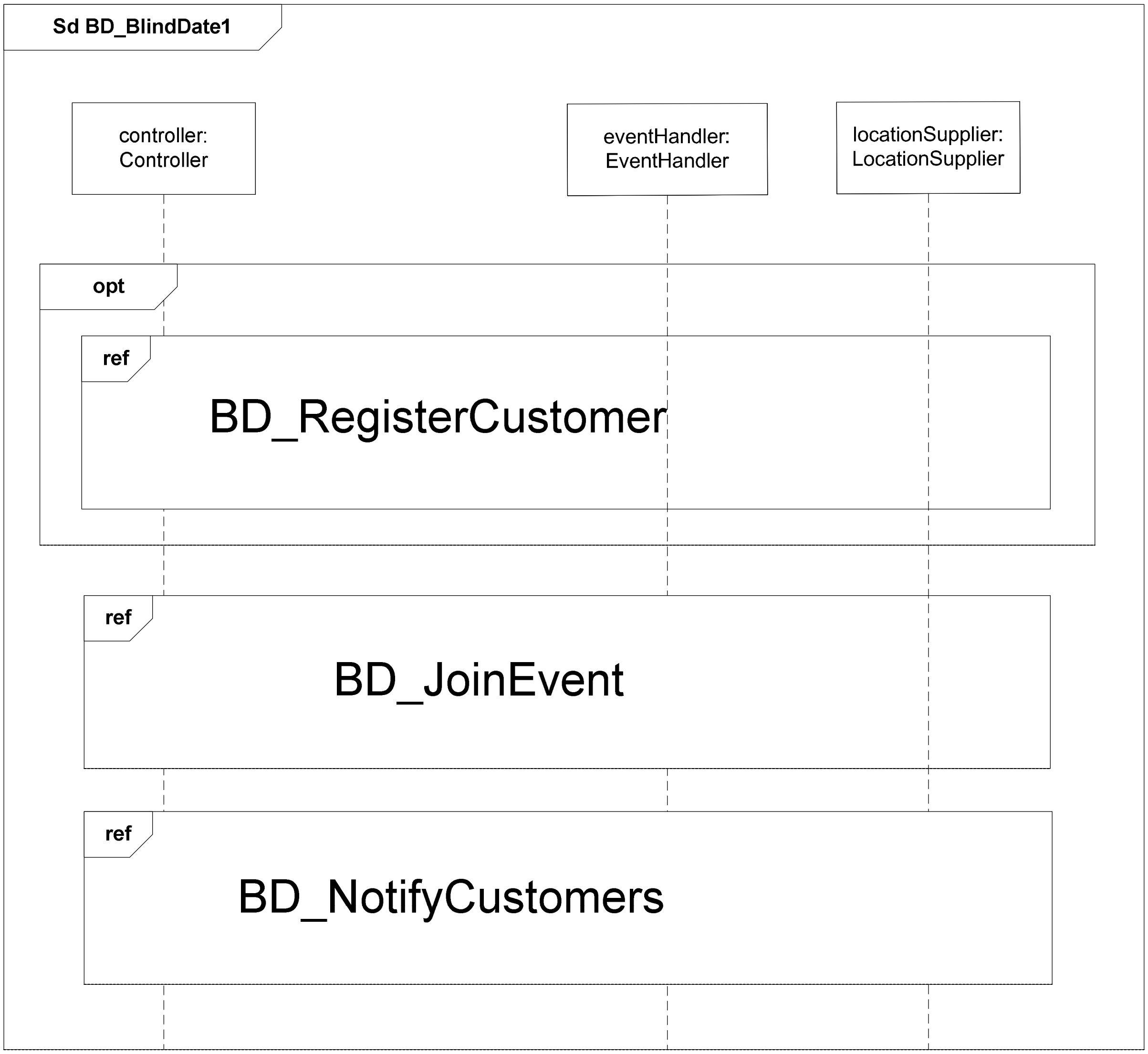

Controller
fromDynTraf
FromEnv
fromLocationSupplier
fromEventHandler
fromOutside
controllersm : ControllerSM
toEnv
toEventHandler
toSession
fromCS_SM
toEnv
toLocationSupplier
toEventHandler
toDynTraf
toEnv
toLocSup
toEventH
toDynTraf
session : Session [*]


EventHandler
fromLocationSupplier
fromEnv
eventhandlersm : EventHandlerSM
fromContoller
toEvent : RouterMediator
fromEventSM
toLocationSupplier
toLocationSup
event : Event [*]
toController
toController

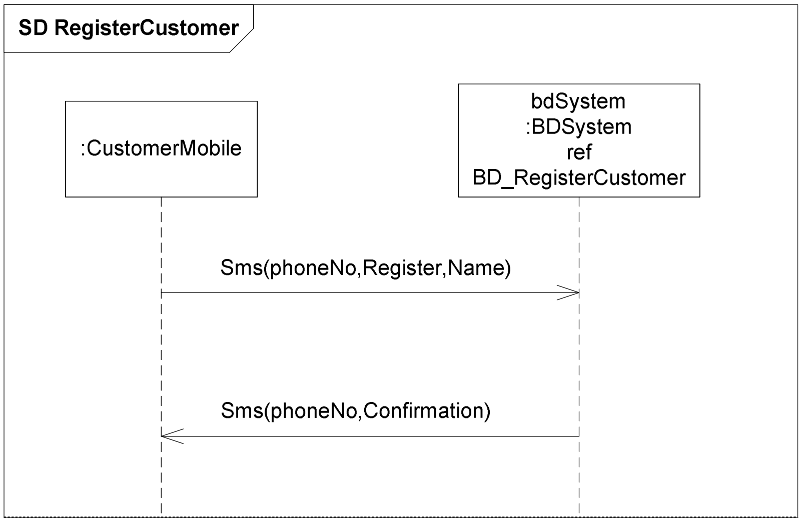
BlindDate1 sequence diagram



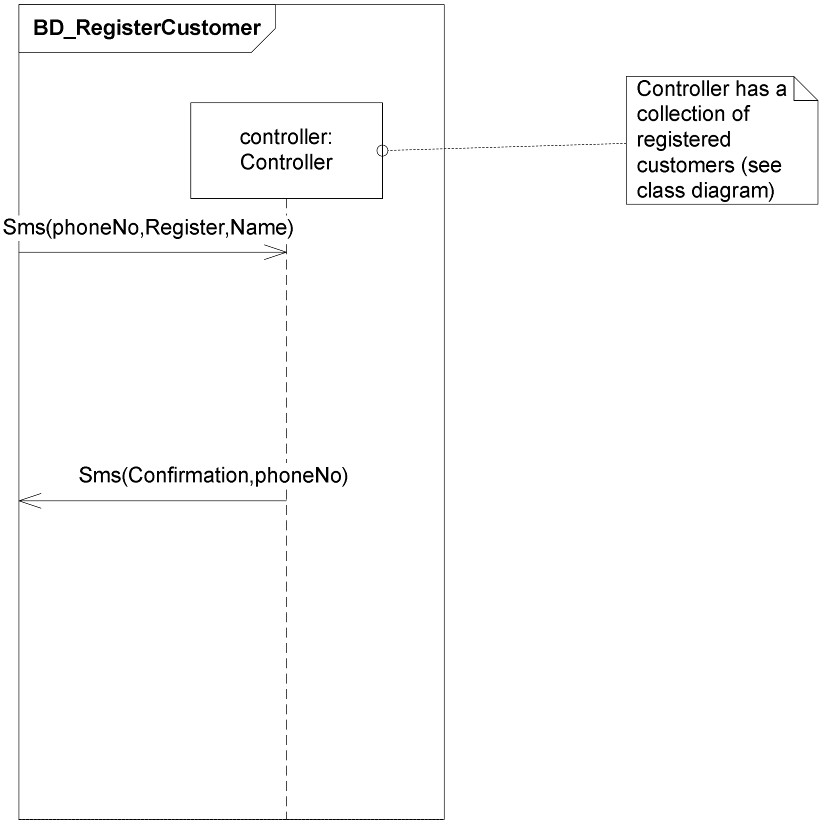
BD_BlindDate1 sequence diagram



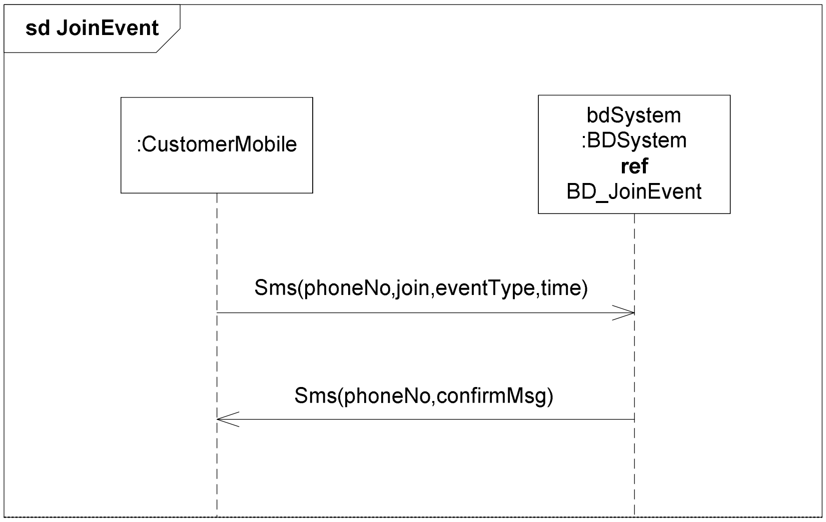
RegisterCustomer sequence diagram



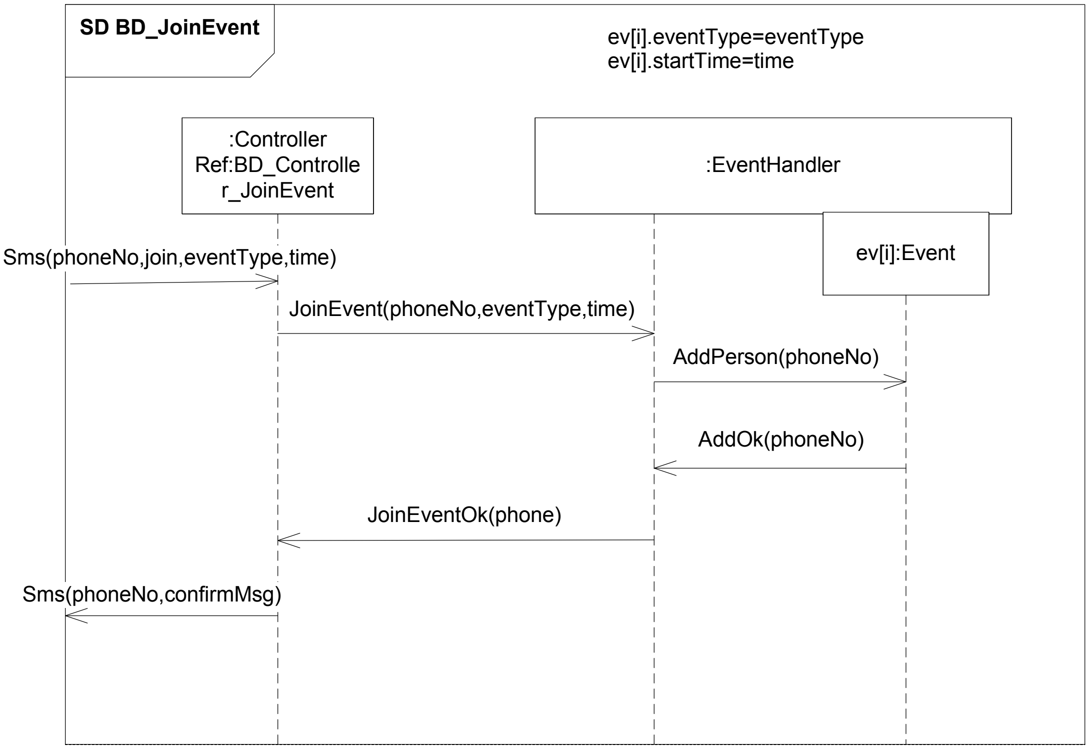
BD_RegisterCustomer sequence diagram


BD_RegisterCustomer
controller:
Controller
Controller has a collection of registered customers (see class diagram)
Sms(phoneNo,Register,Name)
Sms(Confirmation,phoneNo)

JoinEvent sequence diagram

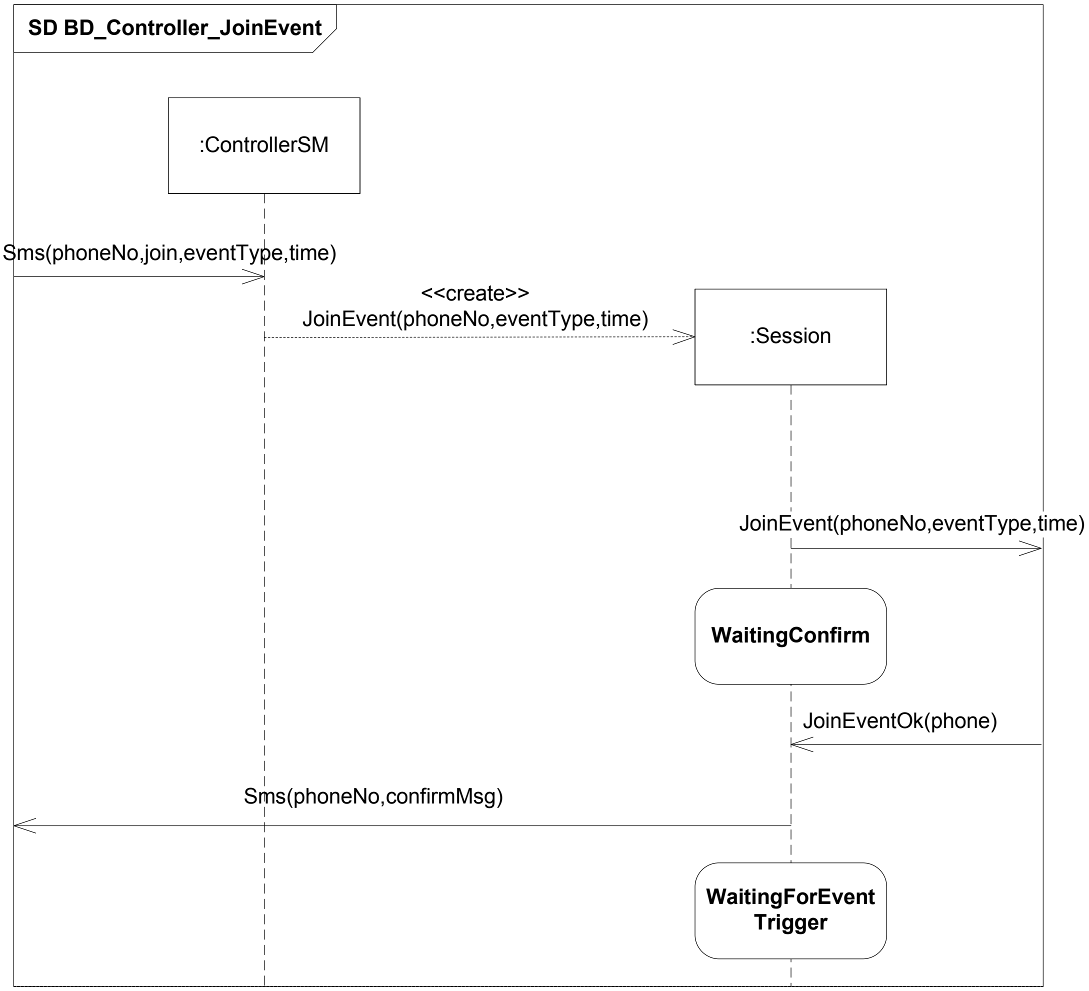


BD_JoinEvent sequence diagram



BD_Controller_JoinEvent sequence diagram

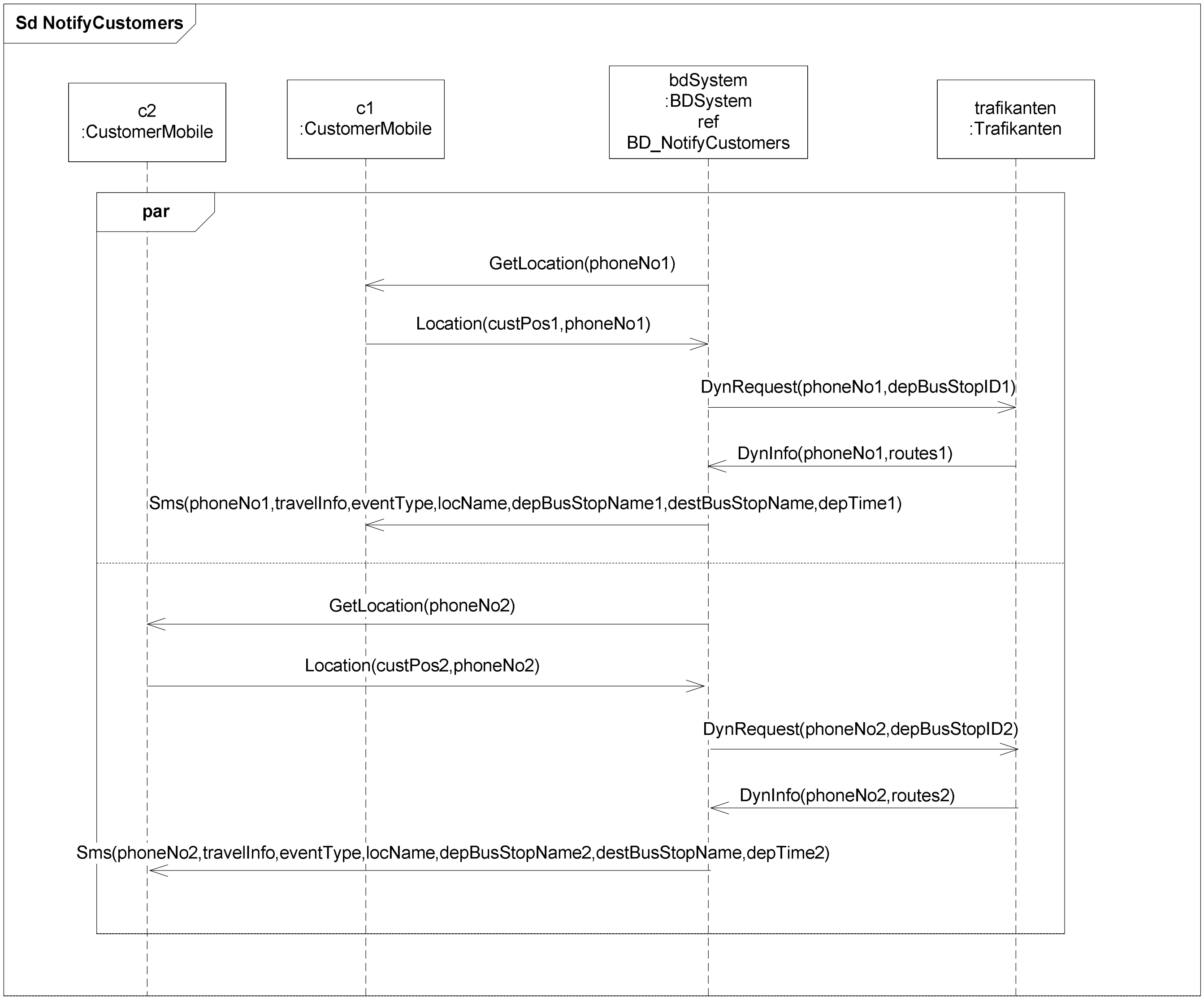
When a customer wants to join a certain event, she/he sends an SMS message with keyword “join” and information about the event type and time. A session is then created to handle the communication with the customer related to this particular event. The session is alive until the notification with travel information is sent (shown in BD_Controller_NCust).


SD BD_Controller_JoinEvent
:ControllerSM
Sms(phoneNo,join,eventType,time)
<<create>>
JoinEvent(phoneNo,eventType,time)
:Session
JoinEvent(phoneNo,eventType,time)
WaitingConfirm
JoinEventOk(phone)
Sms(phoneNo,confirmMsg)
WaitingForEvent Trigger

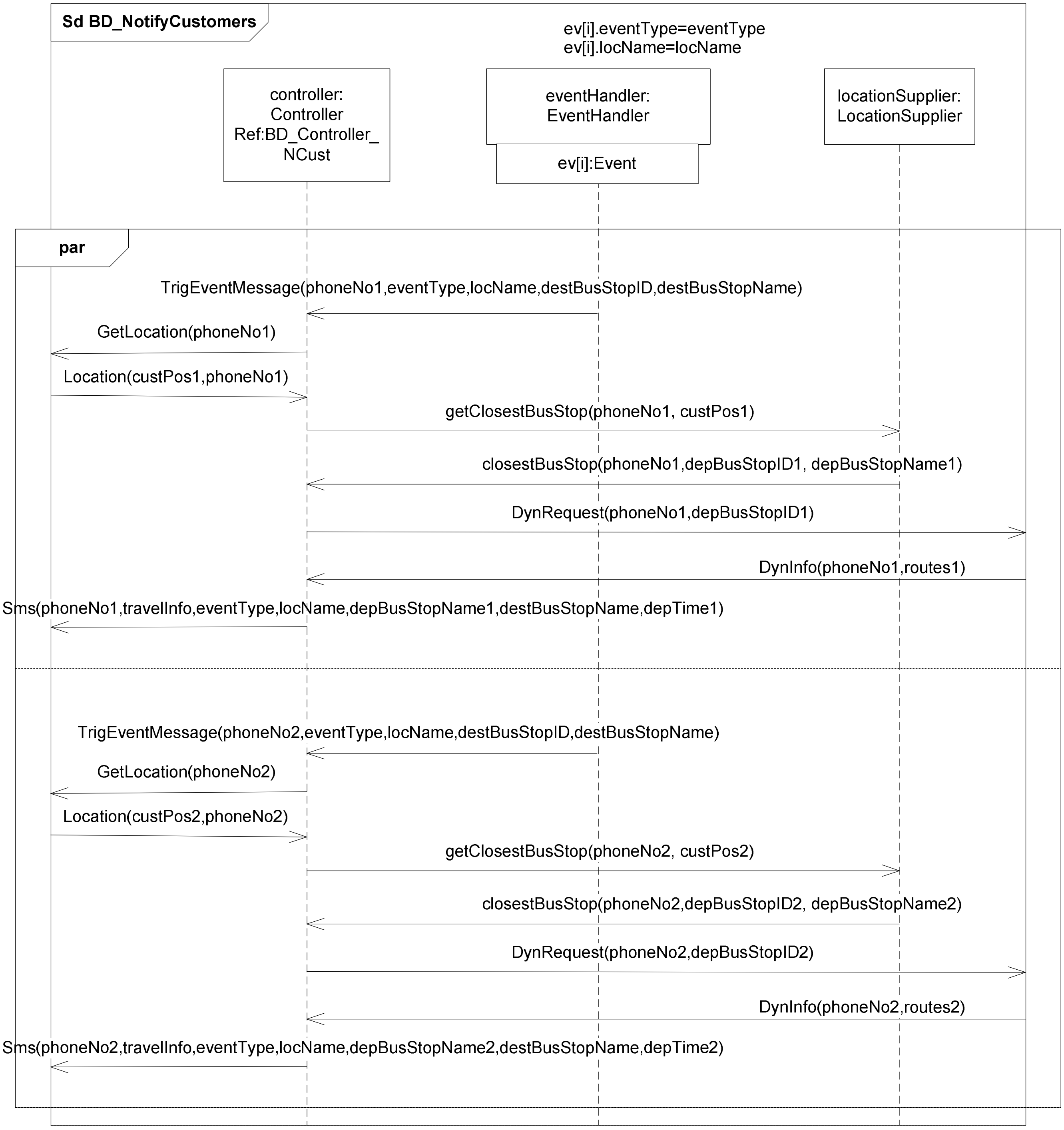
NotifyCustomers sequence diagram

At an appropriate time before an event the system collects information about the location for each participating customer, obtains travel information from Trafikanten, and sends travel advice to the customer.

The number of participants can differ between events; we have chosen to show the case with two participants for the same specific event. Note that the parameters eventType and locName are identical in the two operands of the par operator. This convention throughout the specification.

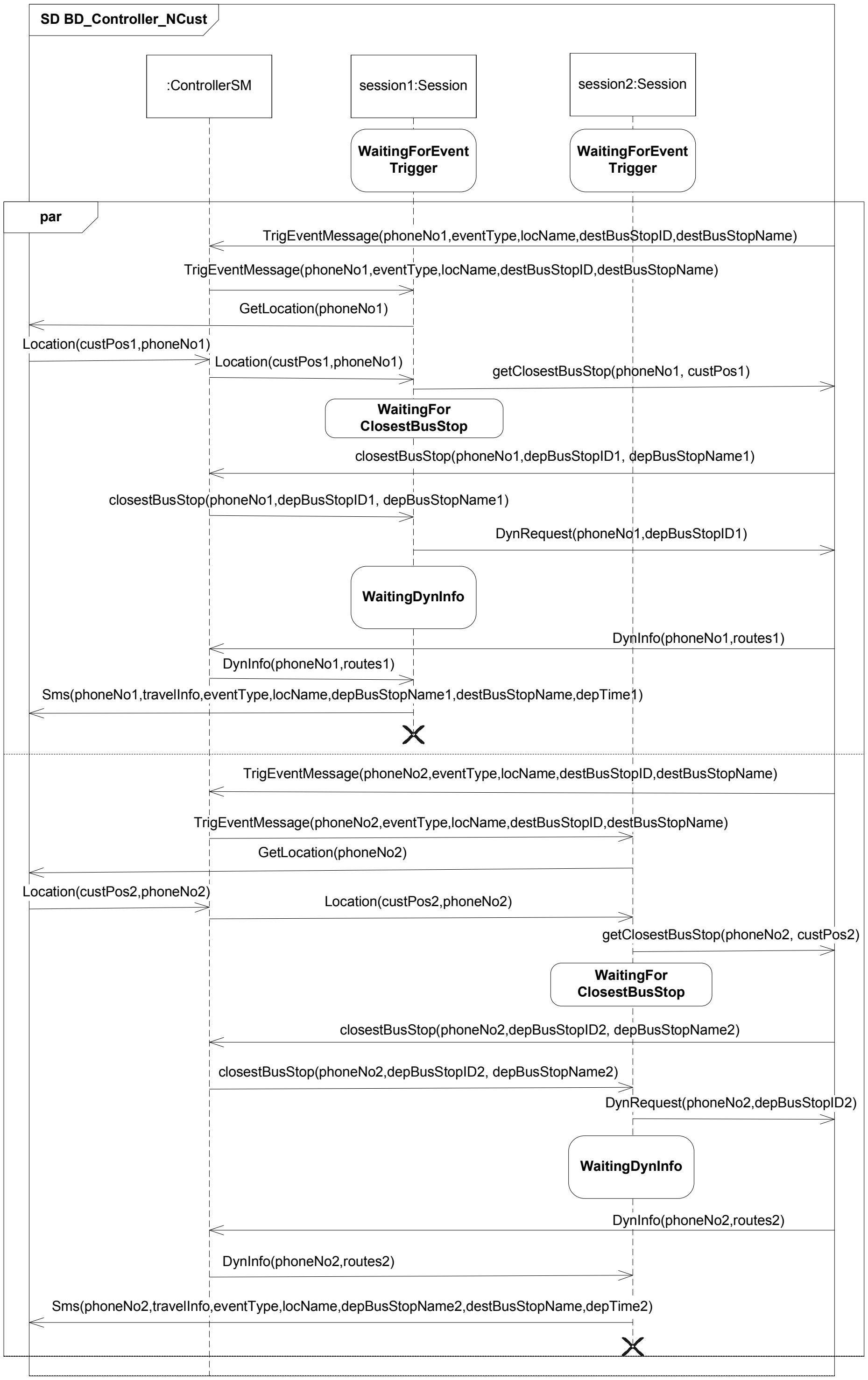

Sd NotifyCustomers
c2
:CustomerMobile
c1
:CustomerMobile
bdSystem
:BDSystem
ref
BD_NotifyCustomers
trafikanten
:Trafikanten
par
GetLocation(phoneNo1)
Location(custPos1,phoneNo1)
DynRequest(phoneNo1,depBusStopID1)
DynInfo(phoneNo1,routes1)
Sms(phoneNo1,travelInfo,eventType,locName,depBusStopName1,destBusStopName,depTime1)
GetLocation(phoneNo2)
Location(custPos2,phoneNo2)
DynRequest(phoneNo2,depBusStopID2)
DynInfo(phoneNo2,routes2)
Sms(phoneNo2,travelInfo,eventType,locName,depBusStopName2,destBusStopName,depTime2)

BD_NotifyCustomers sequence diagram


Sd BD_NotifyCustomers
ev[i].eventType=eventType
ev[i].locName=locName
controller: Controller Ref:BD_Controller_ NCust
eventHandler: EventHandler
ev[i]:Event
locationSupplier: LocationSupplier
par
TrigEventMessage(phoneNo1,eventType,locName,destBusStopID,destBusStopName)
GetLocation(phoneNo1)
Location(custPos1,phoneNo1)
getClosestBusStop(phoneNo1, custPos1)
closestBusStop(phoneNo1,depBusStopID1, depBusStopName1)
DynRequest(phoneNo1,depBusStopID1)
DynInfo(phoneNo1,routes1)
Sms(phoneNo1,travelInfo,eventType,locName,depBusStopName1,destBusStopName,depTime1)
TrigEventMessage(phoneNo2,eventType,locName,destBusStopID,destBusStopName)
GetLocation(phoneNo2)
Location(custPos2,phoneNo2)
getClosestBusStop(phoneNo2, custPos2)
closestBusStop(phoneNo2,depBusStopID2, depBusStopName2)
DynRequest(phoneNo2,depBusStopID2)
DynInfo(phoneNo2,routes2)
Sms(phoneNo2,travelInfo,eventType,locName,depBusStopName2,destBusStopName,depTime2)

BD_Controller_NCust sequence diagram

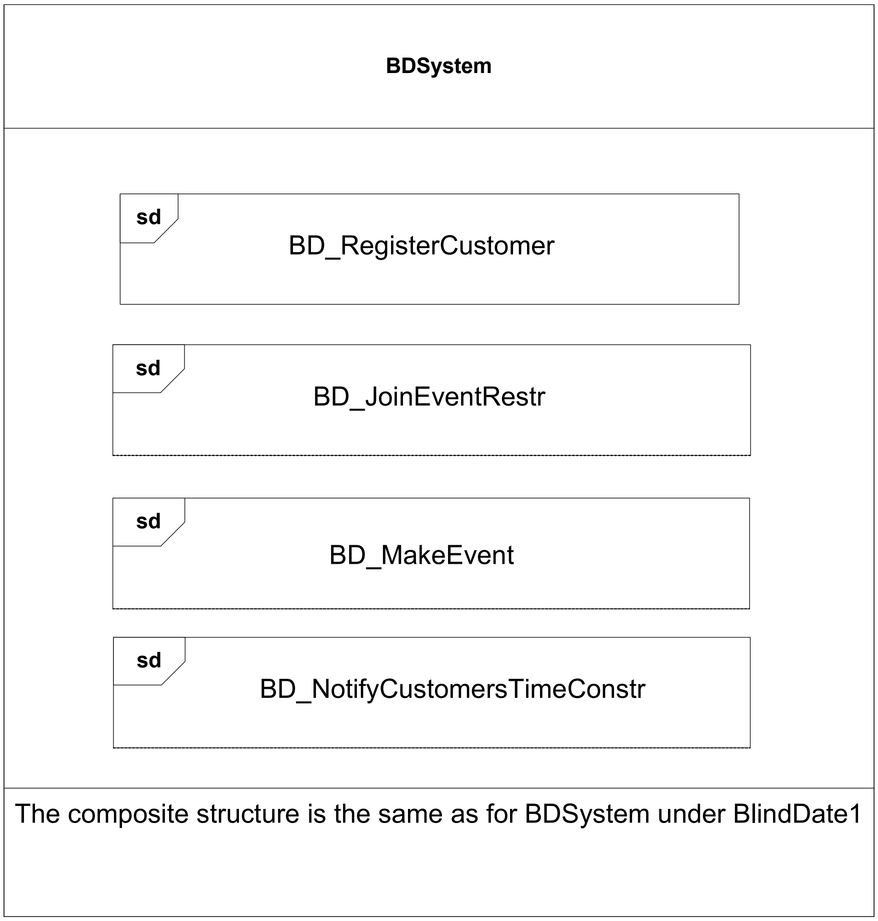
In this diagram suggestions for some state names are included. Not all states are necessarily shown. Notice that the initial state names are identical to those in BD_Controller_JoinEvent.


SD BD_Controller_NCust
:ControllerSM
session1:Session
session2:Session
WaitingForEvent Trigger
WaitingForEvent Trigger
par
TrigEventMessage(phoneNo1,eventType,locName,destBusStopID,destBusStopName)
TrigEventMessage(phoneNo1,eventType,locName,destBusStopID,destBusStopName)
GetLocation(phoneNo1)
Location(custPos1,phoneNo1)
Location(custPos1,phoneNo1)
getClosestBusStop(phoneNo1, custPos1)
WaitingFor ClosestBusStop
closestBusStop(phoneNo1,depBusStopID1, depBusStopName1)
closestBusStop(phoneNo1,depBusStopID1, depBusStopName1)
DynRequest(phoneNo1,depBusStopID1)
WaitingDynInfo
DynInfo(phoneNo1,routes1)
DynInfo(phoneNo1,routes1)
Sms(phoneNo1,travelInfo,eventType,locName,depBusStopName1,destBusStopName,depTime1)
TrigEventMessage(phoneNo2,eventType,locName,destBusStopID,destBusStopName)
TrigEventMessage(phoneNo2,eventType,locName,destBusStopID,destBusStopName)
GetLocation(phoneNo2)
Location(custPos2,phoneNo2)
Location(custPos2,phoneNo2)
getClosestBusStop(phoneNo2, custPos2)
WaitingFor ClosestBusStop
closestBusStop(phoneNo2,depBusStopID2, depBusStopName2)
closestBusStop(phoneNo2,depBusStopID2, depBusStopName2)
DynRequest(phoneNo2,depBusStopID2)
WaitingDynInfo
DynInfo(phoneNo2,routes2)
DynInfo(phoneNo2,routes2)
Sms(phoneNo2,travelInfo,eventType,locName,depBusStopName2,destBusStopName,depTime2)

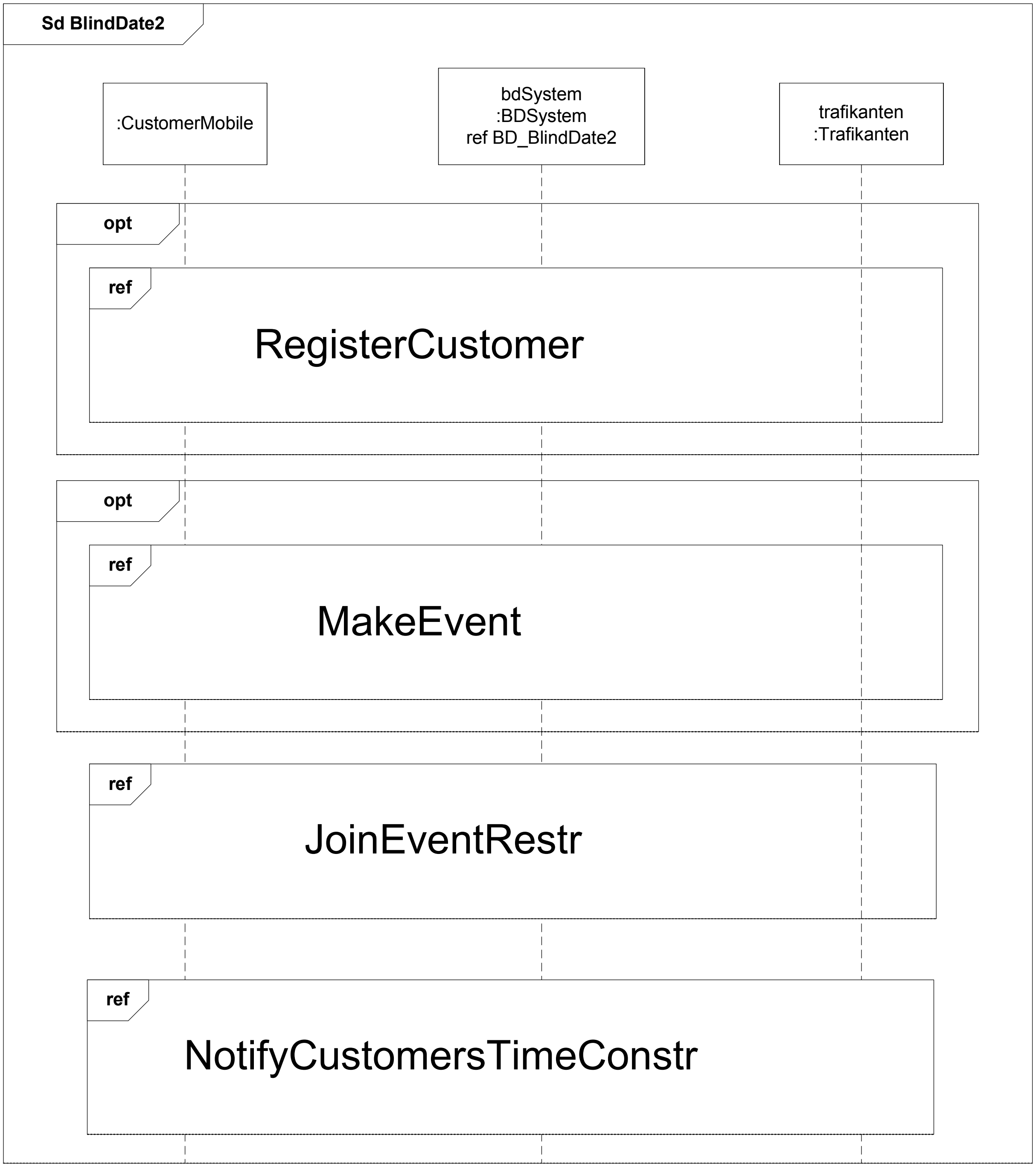
BlindDate2

This section describes the diagrams relevant for BlindDate2. Diagrams that are identical to BlindDate1 are not repeated.

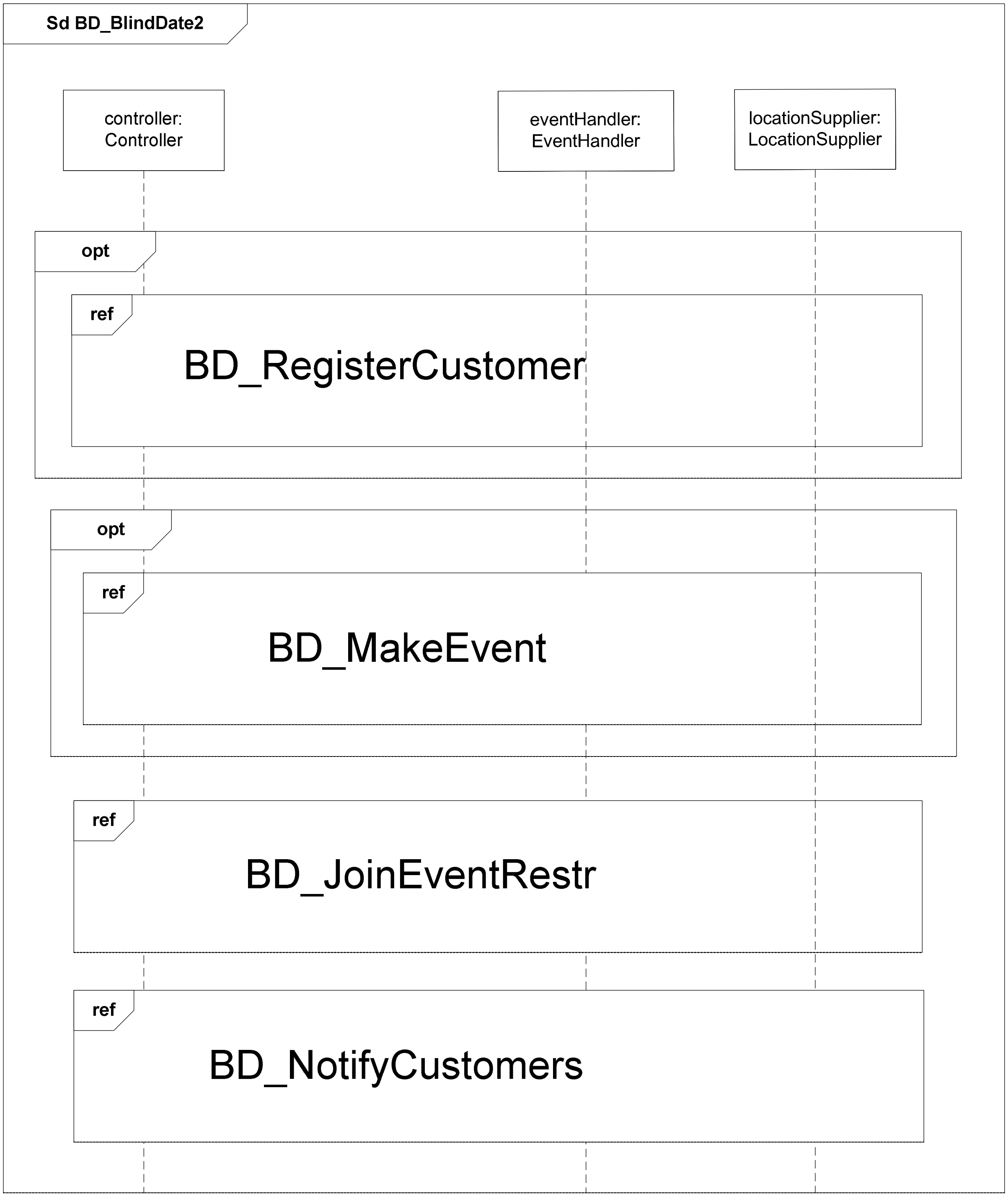
BDSystem structure


BDSystem
sd
BD_RegisterCustomer
sd
BD_JoinEventRestr
sd
BD_MakeEvent
sd
BD_NotifyCustomersTimeConstr
The composite structure is the same as for BDSystem under BlindDate1

BlindDate2 sequence diagram

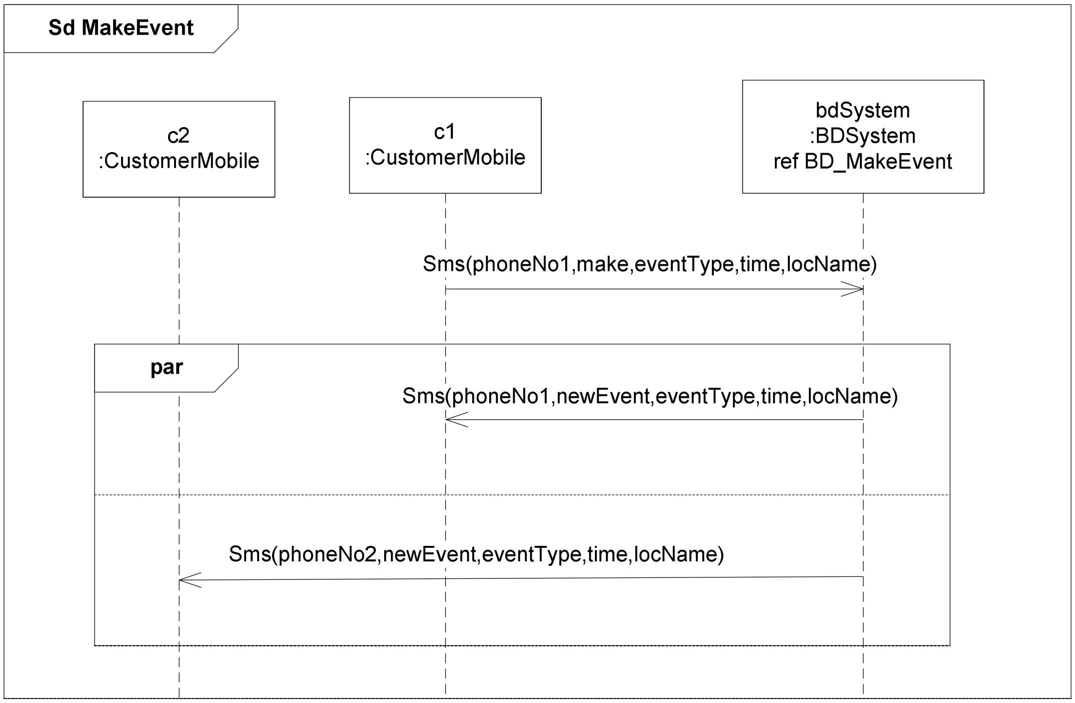


BD_BlindDate2 sequence diagram

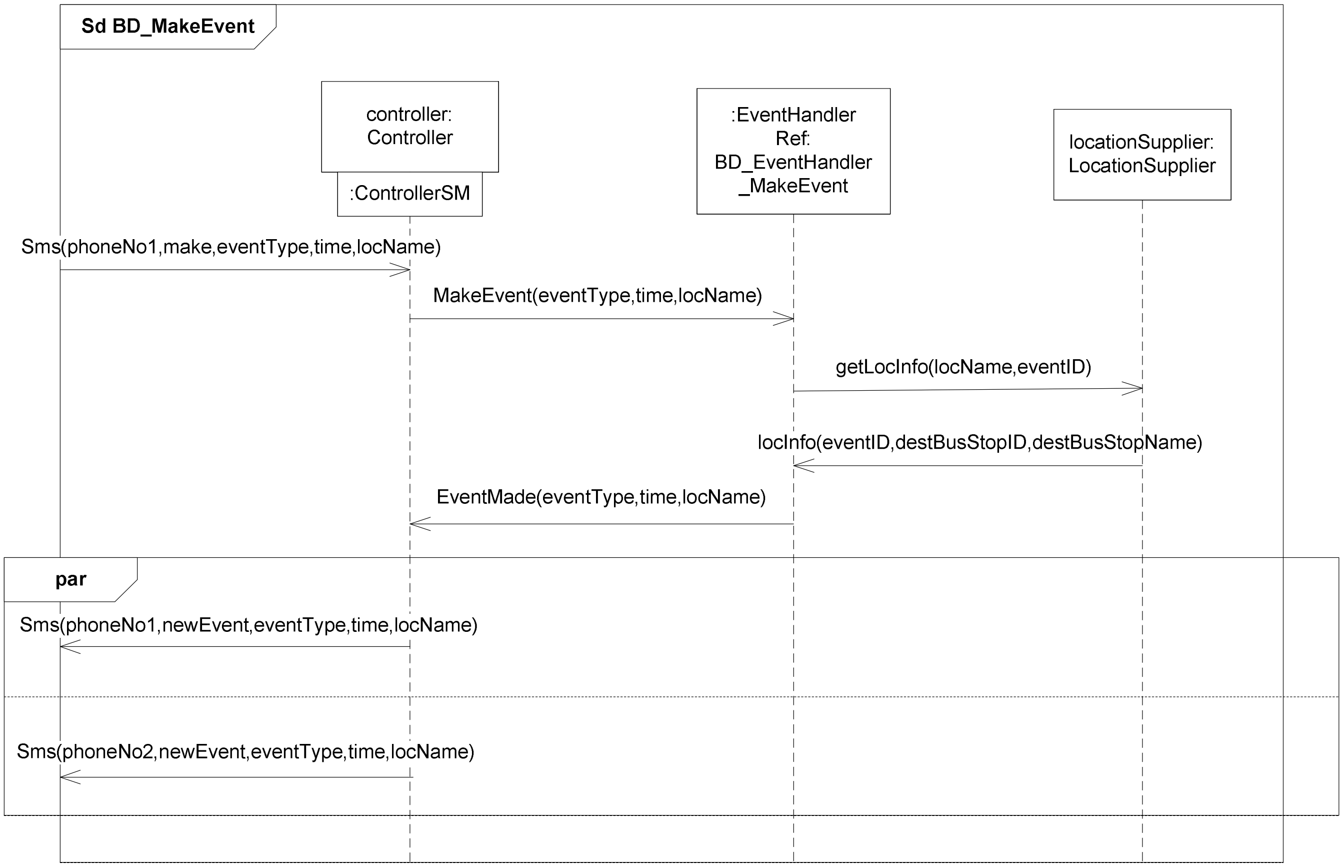


MakeEvent sequence diagram

This diagram shows the possibility for a customer to initiate the creation of an event. After the event has been created, a message is broadcast to all customers of the system. Note that the customer that initiates the event is not automatically registered as a participant.
Events may still be generated by the system at startup.

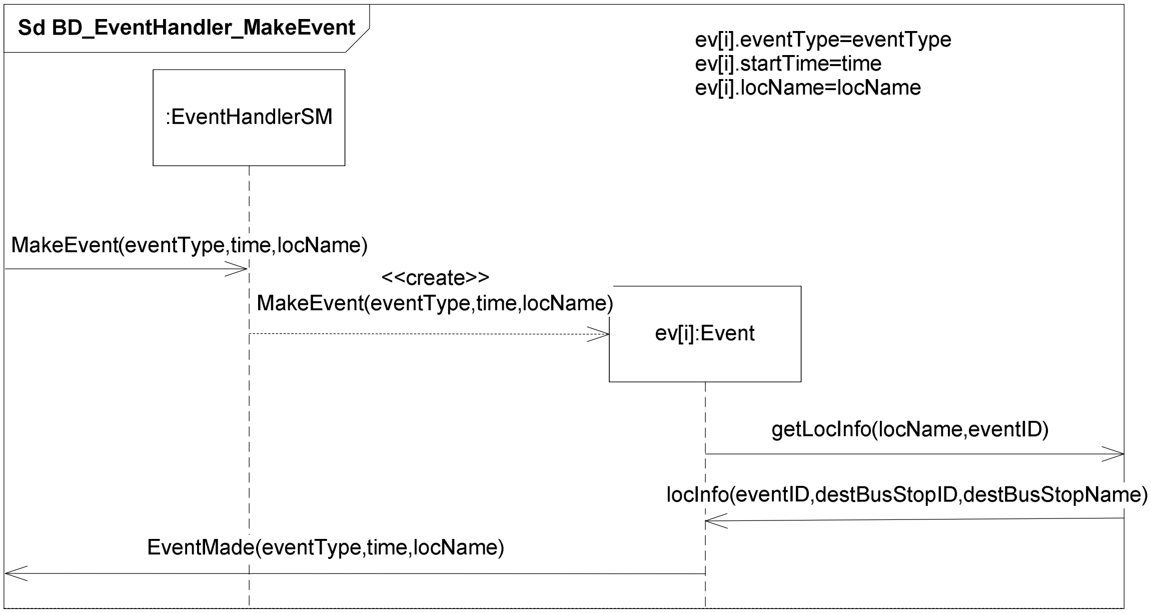

Sd MakeEvent
c2
:CustomerMobile
c1
:CustomerMobile
bdSystem
:BDSystem
ref BD_MakeEvent
Sms(phoneNo1,make,eventType,time,locName)
par
Sms(phoneNo1,newEvent,eventType,time,locName)
Sms(phoneNo2,newEvent,eventType,time,locName)

BD_MakeEvent sequence diagram


Sd BD_MakeEvent
controller:
Controller
:ControllerSM
:EventHandler
Ref:
BD_EventHandler
_MakeEvent
locationSupplier:
LocationSupplier
Sms(phoneNo1,make,eventType,time,locName)
MakeEvent(eventType,time,locName)
getLocInfo(locName,eventID)
locInfo(eventID,destBusStopID,destBusStopName)
EventMade(eventType,time,locName)
par
Sms(phoneNo1,newEvent,eventType,time,locName)
Sms(phoneNo2,newEvent,eventType,time,locName)

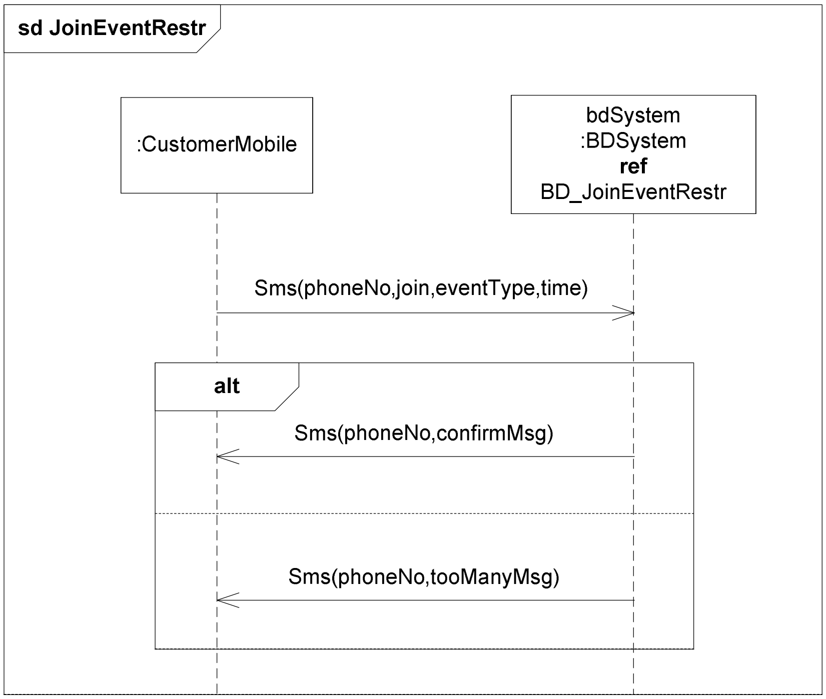
BD_EventHandler_MakeEvent sequence diagram

Events in BlindDate2 (and also BlindDate1) are assumed to have information about their location and nearest bus stop. Therefore an event requests information about the nearest bus stop immediately after creation.

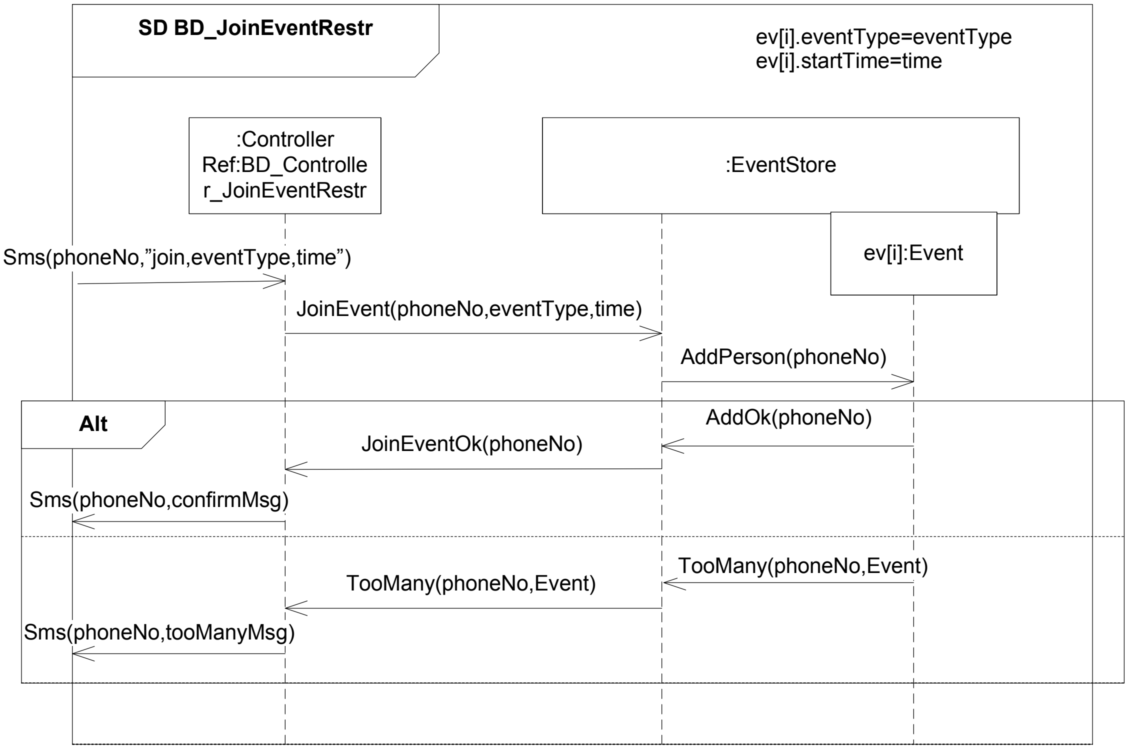

Sd BD_EventHandler_MakeEvent
ev[i].eventType=eventType
ev[i].startTime=time
ev[i].locName=locName
:EventHandlerSM
MakeEvent(eventType,time,locName)
<<create>>
MakeEvent(eventType,time,locName)
ev[i]:Event
getLocInfo(locName,eventID)
locInfo(eventID,destBusStopID,destBusStopName)
EventMade(eventType,time,locName)

JoinEventRestr sequence diagram

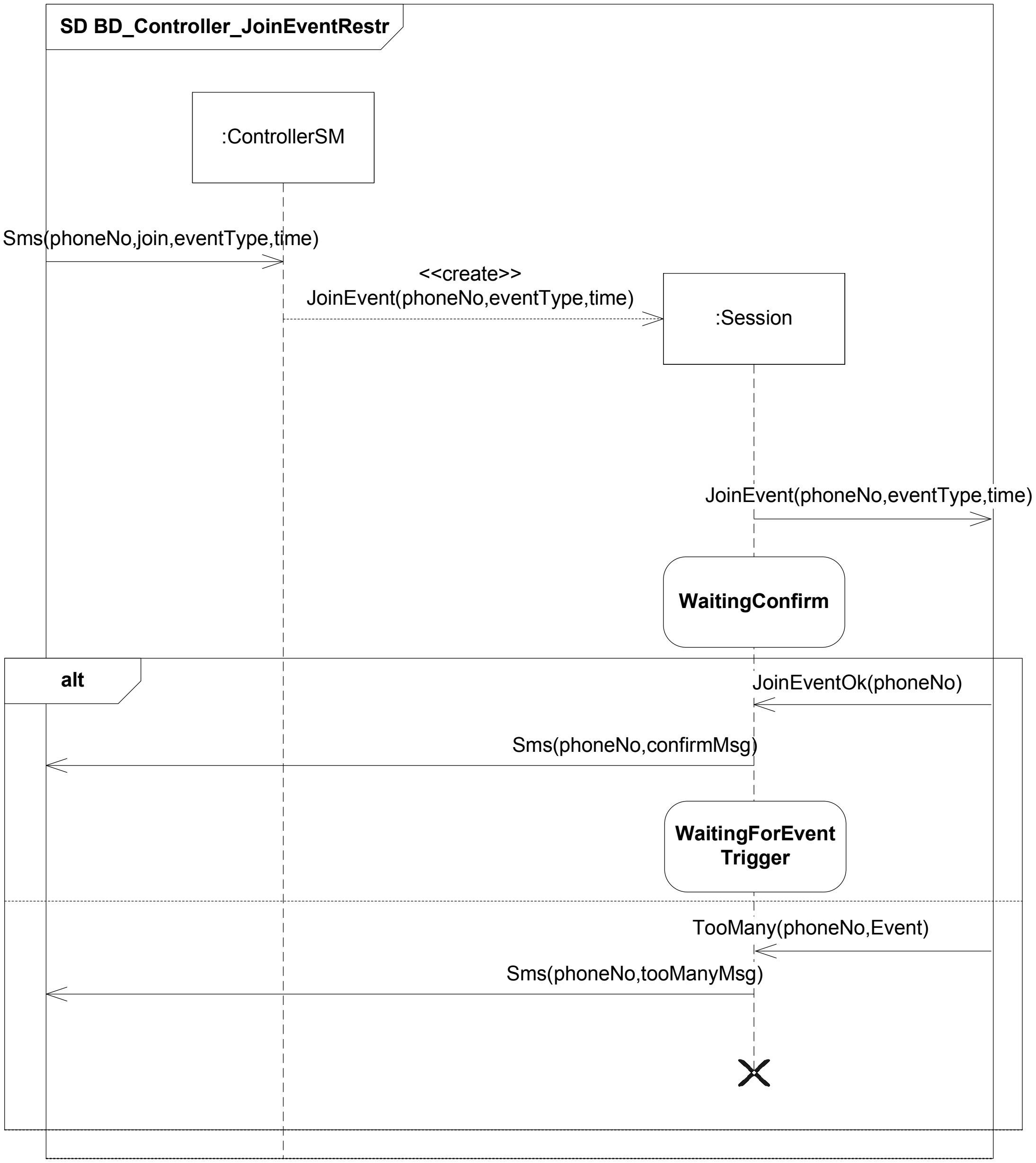
In this version the system may reject a customer; this happens if too many customers have registered as a participant for the event.



BD_JoinEventRestr sequence diagram

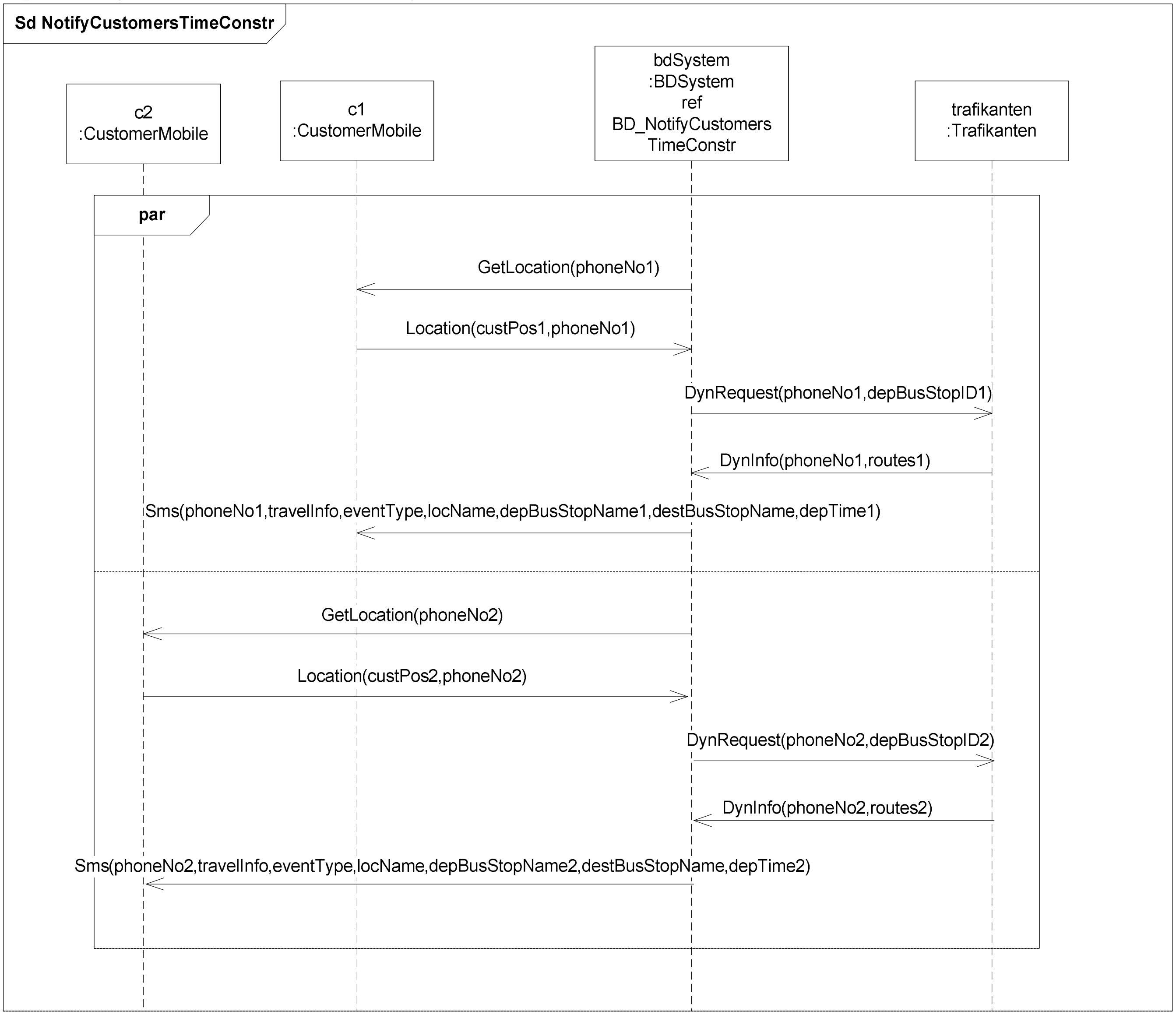


BD_Controller_JoinEventRestr sequence diagram



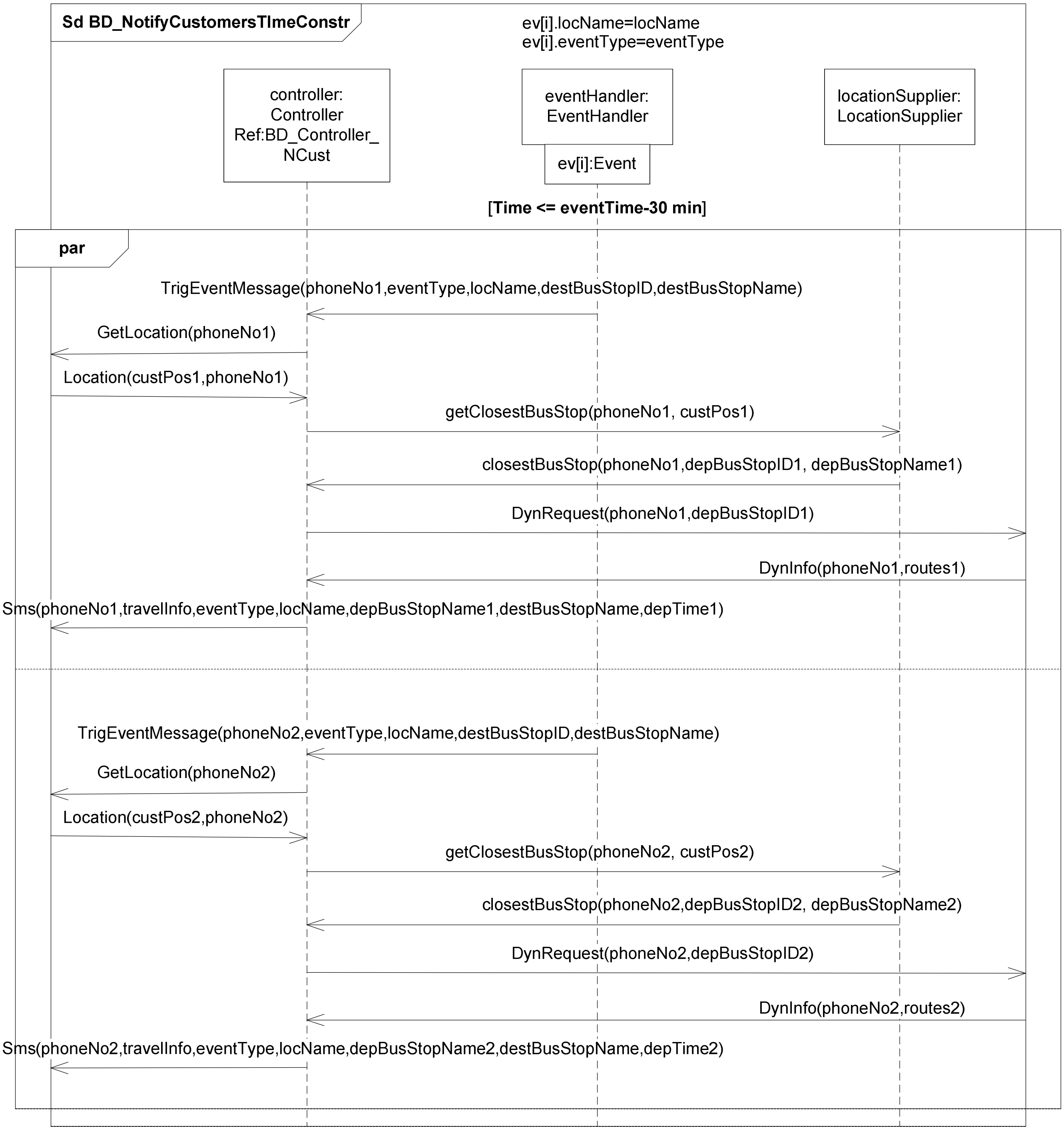
NotifyCustomersTimeConstr sequence diagram

This diagram is the same as NotifyCustomers except that the reference to BD_NotifyCustomers is replaced by a reference to BD_NotifyCustomersTimeConstr


Sd NotifyCustomersTimeConstr
c2
:CustomerMobile
c1
:CustomerMobile
bdSystem
:BDSystem
ref
BD_NotifyCustomers
TimeConstr
trafikanten
:Trafikanten
par
GetLocation(phoneNo1)
Location(custPos1,phoneNo1)
DynRequest(phoneNo1,depBusStopID1)
DynInfo(phoneNo1,routes1)
Sms(phoneNo1,travelInfo,eventType,locName,depBusStopName1,destBusStopName,depTime1)
GetLocation(phoneNo2)
Location(custPos2,phoneNo2)
DynRequest(phoneNo2,depBusStopID2)
DynInfo(phoneNo2,routes2)
Sms(phoneNo2,travelInfo,eventType,locName,depBusStopName2,destBusStopName,depTime2)

BD_NotifyCustomersTimeConstr sequence diagram

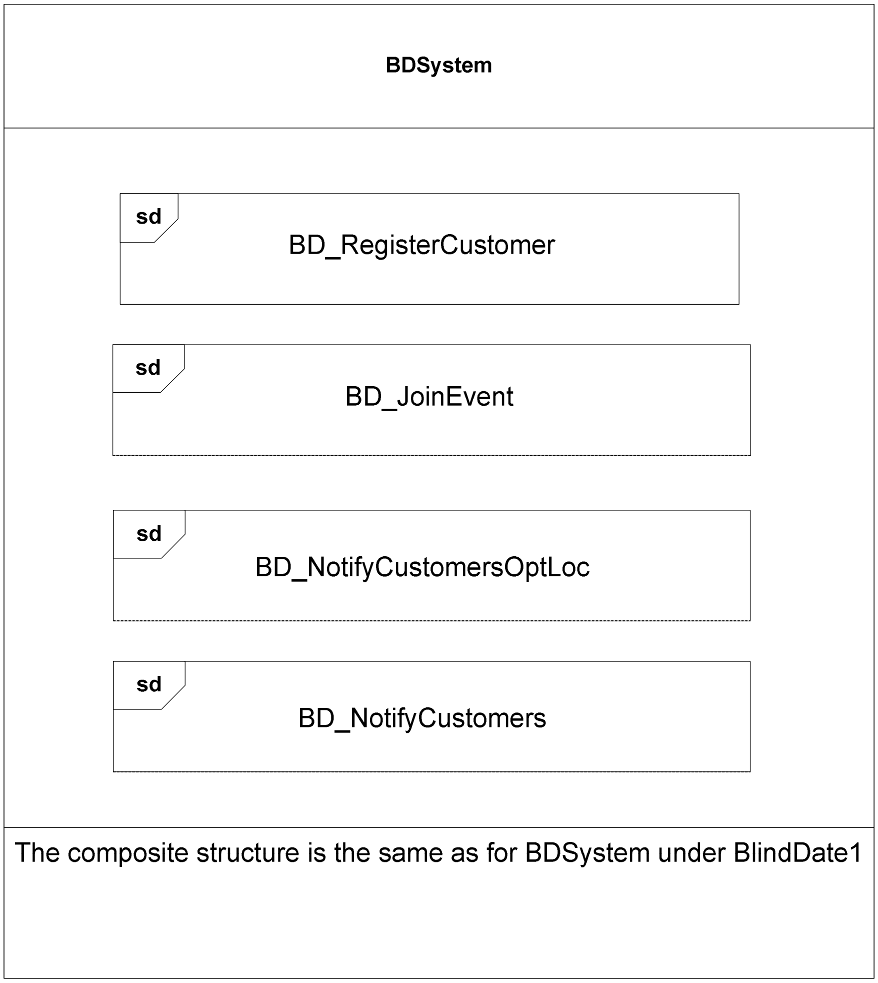
The only difference between this diagram and BD_ NotifyCustomers is that a time constraint has been added.


Sd BD_NotifyCustomersTImeConstr
ev[i].locName=locName
ev[i].eventType=eventType
controller: Controller Ref:BD_Controller_ NCust
eventHandler: EventHandler
ev[i]:Event
locationSupplier: LocationSupplier
[Time <= eventTime-30 min]
par
TrigEventMessage(phoneNo1,eventType,locName,destBusStopID,destBusStopName)
GetLocation(phoneNo1)
Location(custPos1,phoneNo1)
getClosestBusStop(phoneNo1, custPos1)
closestBusStop(phoneNo1,depBusStopID1, depBusStopName1)
DynRequest(phoneNo1,depBusStopID1)
DynInfo(phoneNo1,routes1)
Sms(phoneNo1,travelInfo,eventType,locName,depBusStopName1,destBusStopName,depTime1)
TrigEventMessage(phoneNo2,eventType,locName,destBusStopID,destBusStopName)
GetLocation(phoneNo2)
Location(custPos2,phoneNo2)
getClosestBusStop(phoneNo2, custPos2)
closestBusStop(phoneNo2,depBusStopID2, depBusStopName2)
DynRequest(phoneNo2,depBusStopID2)
DynInfo(phoneNo2,routes2)
Sms(phoneNo2,travelInfo,eventType,locName,depBusStopName2,destBusStopName,depTime2)

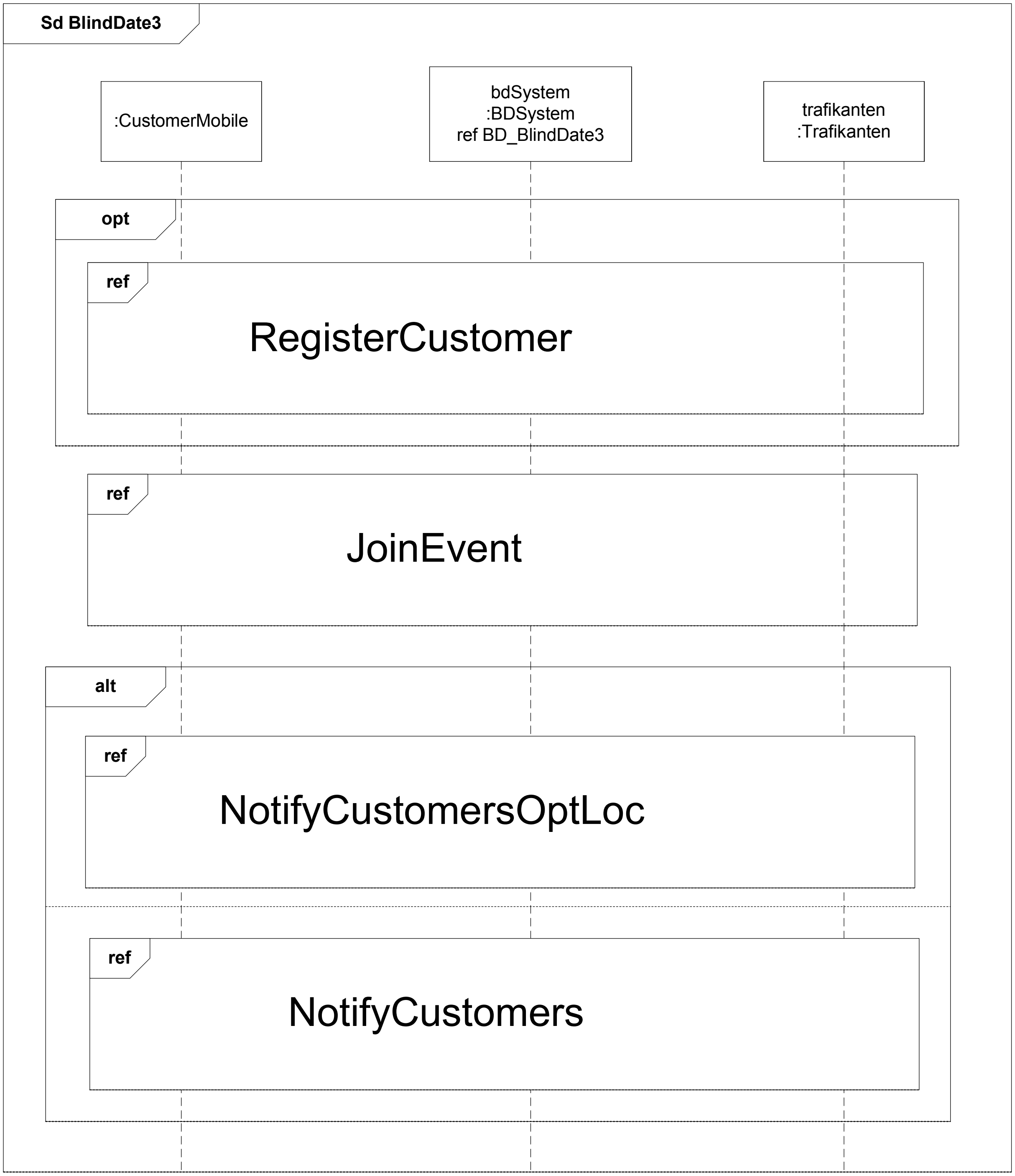
BlindDate3

This section describes the diagrams relevant for BlindDate3. Diagrams that are identical to BlindDate1 are not repeated.

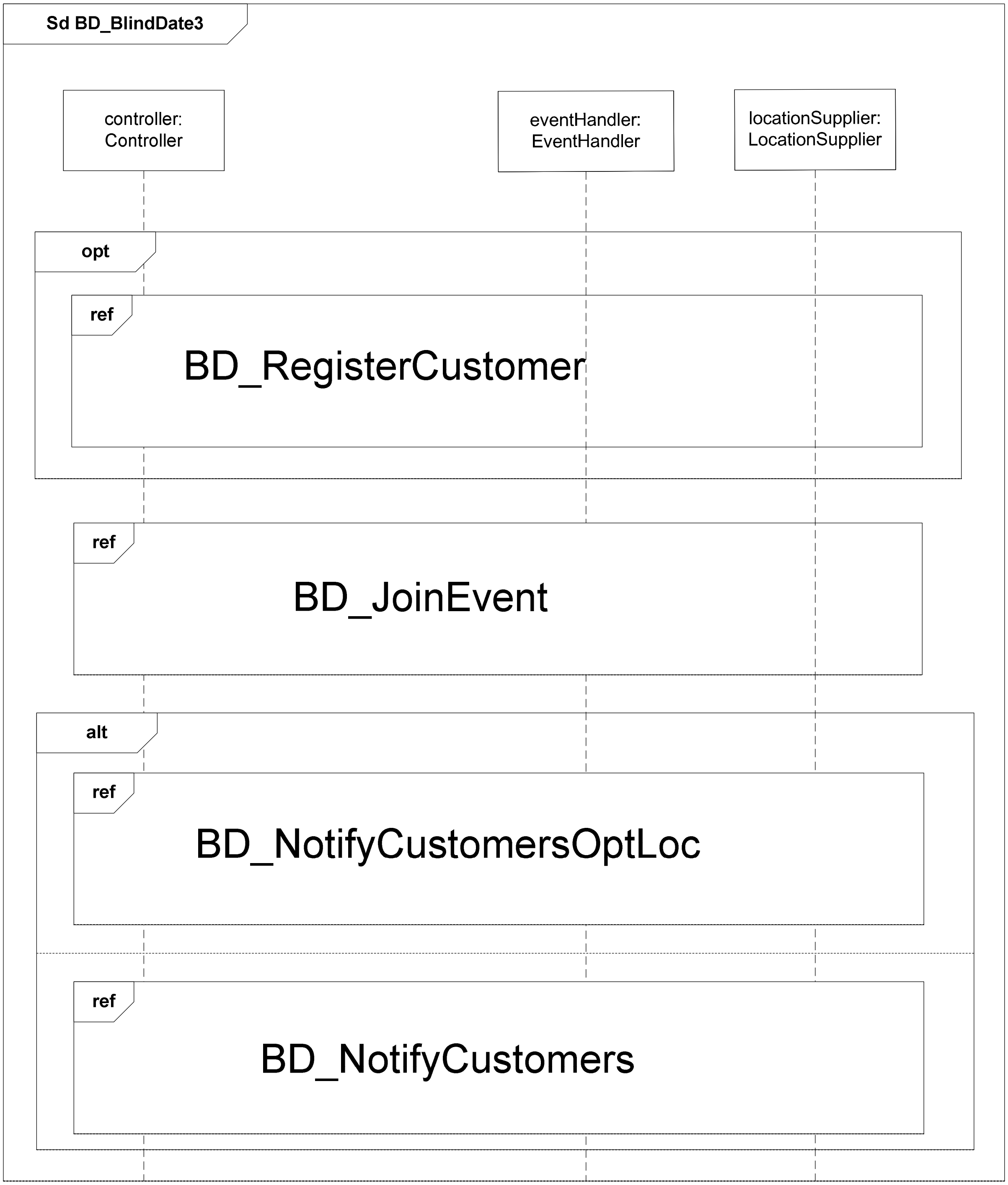
BDSystem structure



BlindDate3 sequence diagram



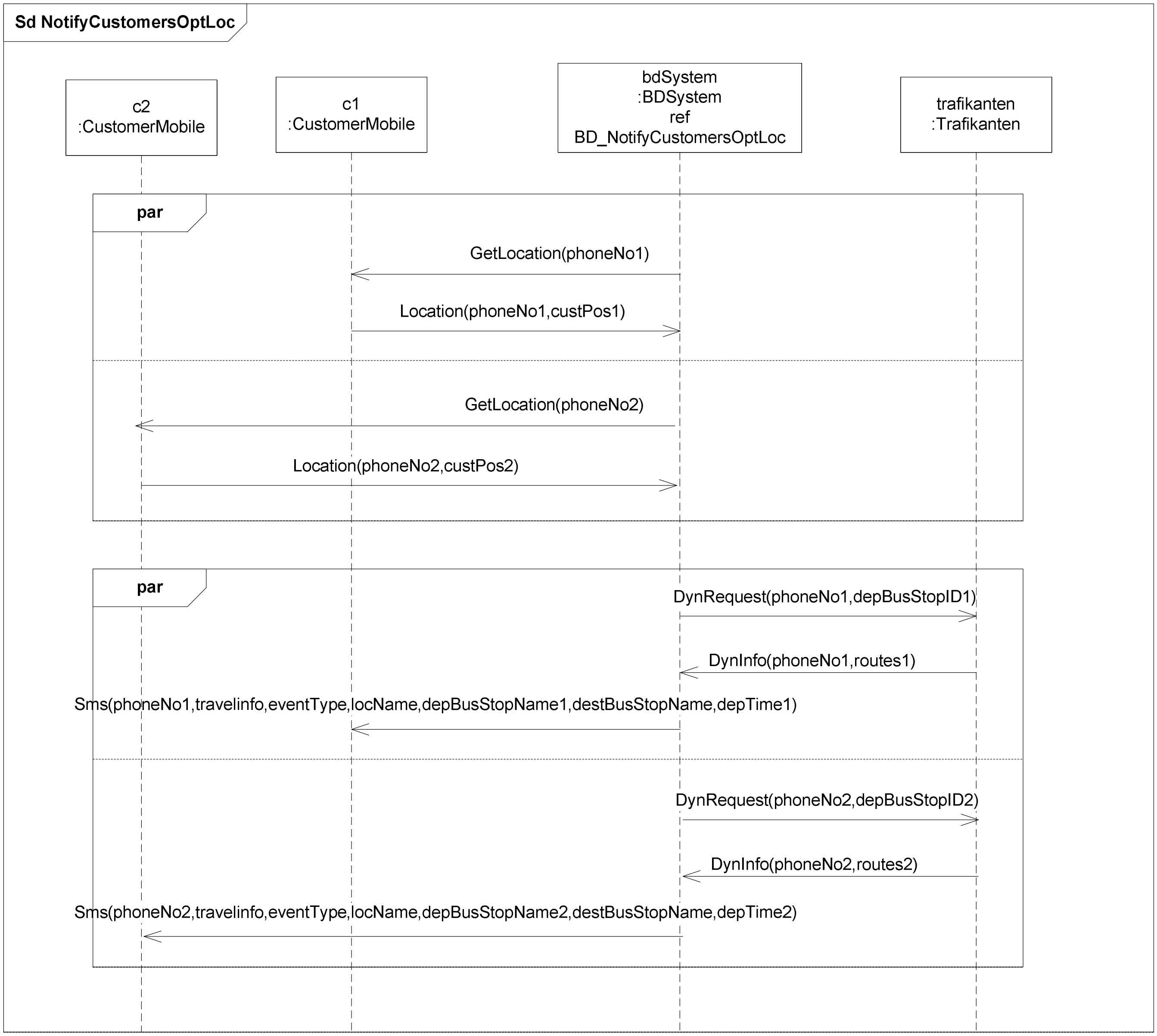
BD_BlindDate3 sequence diagram



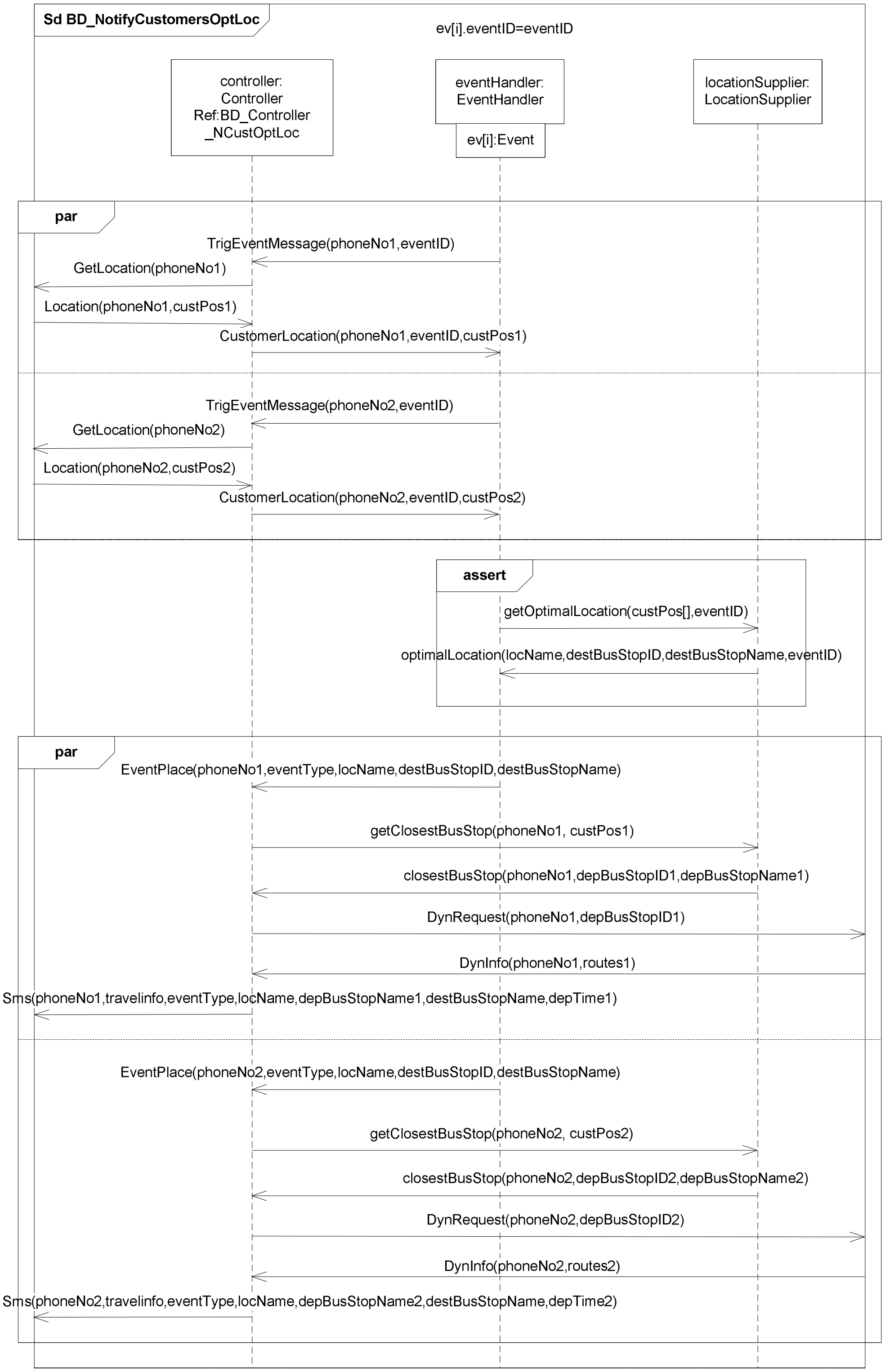
NotifyCustomersOptLoc sequence diagram

This diagram describes the scenario where the location of the event is not fixed until shortly before the event will take place. The location is determined by the location of the participants. Notice that internal behavior of the BDSystem takes place between the two par operators.

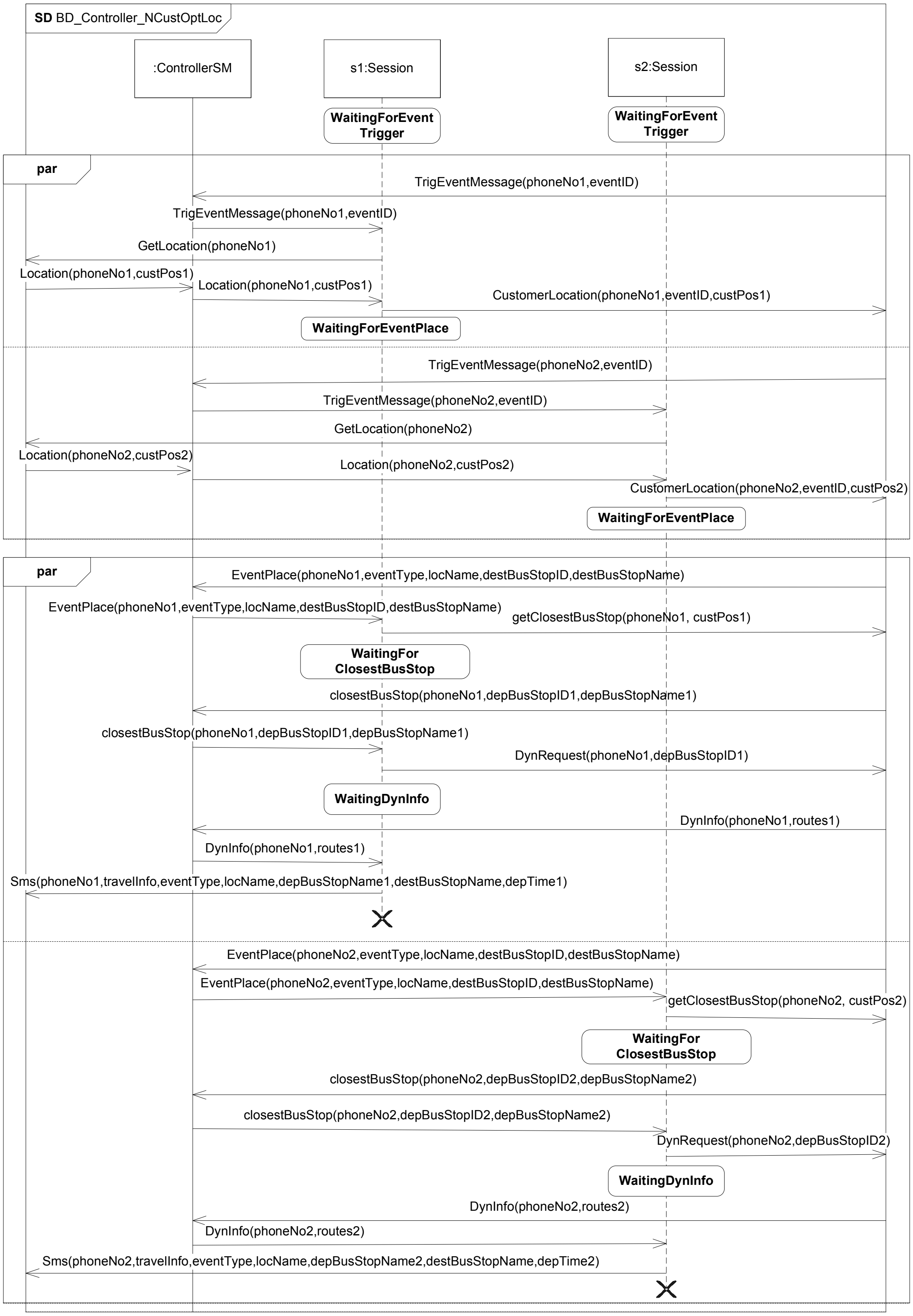
The first par operator describes how information about the position of the participants is collected. The second par operator shows how travel and event information is sent to the customers after the location has been decided. The location of the event is decided by the LocationSupplier based on the positions of the participants.


Sd NotifyCustomersOptLoc
c2 :CustomerMobile
c1 :CustomerMobile
bdSystem :BDSystem ref BD_NotifyCustomersOptLoc
trafikanten :Trafikanten
par
GetLocation(phoneNo1)
Location(phoneNo1,custPos1)
GetLocation(phoneNo2)
Location(phoneNo2,custPos2)
par
DynRequest(phoneNo1,depBusStopID1)
DynInfo(phoneNo1,routes1)
Sms(phoneNo1,travelinfo,eventType,locName,depBusStopName1,destBusStopName,depTime1)
DynRequest(phoneNo2,depBusStopID2)
DynInfo(phoneNo2,routes2)
Sms(phoneNo2,travelinfo,eventType,locName,depBusStopName2,destBusStopName,depTime2)

BD_NotifyCustomersOptLoc sequence diagram


Sd BD_NotifyCustomersOptLoc
ev[i].eventID=eventID
controller: Controller Ref:BD_Controller _NCustOptLoc
eventHandler: EventHandler
ev[i]:Event
locationSupplier: LocationSupplier
par
TrigEventMessage(phoneNo1,eventID)
GetLocation(phoneNo1)
Location(phoneNo1,custPos1)
CustomerLocation(phoneNo1,eventID,custPos1)
TrigEventMessage(phoneNo2,eventID)
GetLocation(phoneNo2)
Location(phoneNo2,custPos2)
CustomerLocation(phoneNo2,eventID,custPos2)
assert
getOptimalLocation(custPos[],eventID)
optimalLocation(locName,destBusStopID,destBusStopName,eventID)
par
EventPlace(phoneNo1,eventType,locName,destBusStopID,destBusStopName)
getClosestBusStop(phoneNo1, custPos1)
closestBusStop(phoneNo1,depBusStopID1,depBusStopName1)
DynRequest(phoneNo1,depBusStopID1)
DynInfo(phoneNo1,routes1)
Sms(phoneNo1,travelinfo,eventType,locName,depBusStopName1,destBusStopName,depTime1)
EventPlace(phoneNo2,eventType,locName,destBusStopID,destBusStopName)
getClosestBusStop(phoneNo2, custPos2)
closestBusStop(phoneNo2,depBusStopID2,depBusStopName2)
DynRequest(phoneNo2,depBusStopID2)
DynInfo(phoneNo2,routes2)
Sms(phoneNo2,travelinfo,eventType,locName,depBusStopName2,destBusStopName,depTime2)

BD_Controller_NCustOptLoc sequence diagram


SD BD_Controller_NCustOptLoc
:ControllerSM
s1:Session
s2:Session
WaitingForEvent Trigger
WaitingForEvent Trigger
par
TrigEventMessage(phoneNo1,eventID)
TrigEventMessage(phoneNo1,eventID)
GetLocation(phoneNo1)
Location(phoneNo1,custPos1)
Location(phoneNo1,custPos1)
CustomerLocation(phoneNo1,eventID,custPos1)
WaitingForEventPlace
TrigEventMessage(phoneNo2,eventID)
TrigEventMessage(phoneNo2,eventID)
GetLocation(phoneNo2)
Location(phoneNo2,custPos2)
Location(phoneNo2,custPos2)
CustomerLocation(phoneNo2,eventID,custPos2)
WaitingForEventPlace
par
EventPlace(phoneNo1,eventType,locName,destBusStopID,destBusStopName)
EventPlace(phoneNo1,eventType,locName,destBusStopID,destBusStopName)
getClosestBusStop(phoneNo1, custPos1)
WaitingFor ClosestBusStop
closestBusStop(phoneNo1,depBusStopID1,depBusStopName1)
closestBusStop(phoneNo1,depBusStopID1,depBusStopName1)
DynRequest(phoneNo1,depBusStopID1)
WaitingDynInfo
DynInfo(phoneNo1,routes1)
DynInfo(phoneNo1,routes1)
Sms(phoneNo1,travelInfo,eventType,locName,depBusStopName1,destBusStopName,depTime1)
EventPlace(phoneNo2,eventType,locName,destBusStopID,destBusStopName)
EventPlace(phoneNo2,eventType,locName,destBusStopID,destBusStopName)
getClosestBusStop(phoneNo2, custPos2)
WaitingFor ClosestBusStop
closestBusStop(phoneNo2,depBusStopID2,depBusStopName2)
closestBusStop(phoneNo2,depBusStopID2,depBusStopName2)
DynRequest(phoneNo2,depBusStopID2)
WaitingDynInfo
DynInfo(phoneNo2,routes2)
DynInfo(phoneNo2,routes2)
Sms(phoneNo2,travelInfo,eventType,locName,depBusStopName2,destBusStopName,depTime2)

Refinement

No xalt operator has been used in any of the specifications in any of the BlindDate systems. Therefore the semantics of all three systems consists of only one interaction obligation.

For any sequence diagram S we use pos(S) to denote the set of positive traces of S, and neg(S) to denote the set of negative traces of S.

The shorthand notation is used for names of sequence diagrams:

RC	=	RegisterCustomer
JE	=	JoinEvent
NC	=	NotifyCustomers
NCTC	=	NotifyCustomersTimeConstr
JER	=	JoinEventRestr
ME	=	MakeEvent
NCO	=	NotifyCustomersOptLoc

Shorthand notations may be prefixed by BD_, so that for example BD_RC refers to BD_RegisterCustomer.

The operator $\succsim$ denotes sequential composition of trace sets and interaction obligations - see 7.2.5. in “STAIRS towards formal design with sequence diagrams” (from now referred to as “the STAIRS paper”).

The semantics of BlindDate1

Since RC is enclosed by an opt operator and RC, JE and NC do not describe any negative behavior, we have that

$$\text{pos(BlindDate1)} = \text{pos(RC)} \succsim \text{pos(JE)} \succsim \text{pos(NC)} \cup \text{pos(JE)} \succsim \text{pos(NC)}$$

$$\text{neg(BlindDate1)} = \emptyset$$

Why BlindDate2 is a refinement of BlindDate1

Since both RE and ME are enclosed by an opt operator and none of the diagrams RC, ME, JER describe negative behavior, we get

$$\begin{aligned}\text{pos(BlindDate2)} = {} & \\ & \text{pos(RC)} \succsim \text{pos(ME)} \succsim \text{pos(JER)} \succsim \text{pos(NCTC)} \cup \\ & \text{pos(RC)} \succsim \text{pos(JER)} \succsim \text{pos(NCTC)} \cup \\ & \text{pos(ME)} \succsim \text{pos(JER)} \succsim \text{pos(NCTC)} \cup \\ & \text{pos(JER)} \succsim \text{pos(NCTC)}\end{aligned}$$

$$\begin{aligned}\text{neg(BlindDate2)} = {} & \\ & \text{pos(RC)} \succsim \text{pos(ME)} \succsim \text{pos(JER)} \succsim \text{neg(NCTC)} \cup\end{aligned}$$

pos(RC) ≿ pos(JER) ≿ neg(NCTC) ∪
pos(ME) ≿ pos(JER) ≿ neg(NCTC) ∪
pos(JER) ≿ neg(NCTC)

Since neg(BlindDate1)=∅, it follows trivially that neg(BlindDate1) ⊆ neg(BlindDate2). We need to show that pos(BlindDate1) ⊆ pos(BlindDate2) ∪ neg(BlindDate2). We do this by showing that

1. pos(JE) ≿ pos(NC)⊆ pos(BlindDate2) ∪ neg(BlindDate2) and
2. pos(RC) ≿ pos(JE) ≿ pos(NC) ⊆ pos(BlindDate2) ∪ neg(BlindDate2).

The only difference between NC and NCTC is that a time restriction has been added, therefore we get

pos(NC) = neg(NCTC) ∪ pos(NCTC).

The only difference between JE and JER is that an alt operator with one new alternative has been added in JER. We therefore get

pos(JE) ⊆ pos(JER).

From this it follows that

pos(JE) ≿ pos(NC)⊆ pos(JER) ≿ pos(NCTC) ∪ pos(JER) ≿ neg(NCTC).

Hence, condition 1 is fulfilled. From this it also follows that

pos(RC) ≿ pos(JE) ≿ pos(NC)⊆
pos(RC) ≿ pos(JER) ≿ pos(NCTC) ∪ pos(RC) ≿ pos(JER) ≿ neg(NCTC)

Therefore condition 2 is also fulfilled, so that BlindDate1⇝ BlindDate2. The time constraint introduced in BlindDate2 ensures that some traces that were positive in BlindDate1 have become negative in BlindDate2, so a narrowing has been performed. At the same time new positive traces has been introduced in BlindDate2 that was inconclusive in BlindDate1, so a supplementing has also been performed.

Why BlindDate3 is a refinement of BlindDate1

In BlindDate3 an alt operator has been introduced that allows a new way of deciding the location of an event in addition to the original version in BlindDate1. No negative behavior has been introduced. Therefore,

pos(BlindDate3) =

pos(RC) ≿ pos(JE) ≿ (pos(NCO) ∪ pos(NC)) ∪
pos(JE) ≿ (pos(NCO) ∪ pos(NC))

neg(BlindDate3)=∅

Since neg(BlindDate1)=∅, it follows trivially that neg(BlindDate1) ⊆ neg(BlindDate3). We need to show that

pos(BlindDate1) ⊆ pos(BlindDate3) ∪ neg(BlindDate3).

Since neg(BlindDate3)=∅, this means that we need to show that

pos(RC) ≿ pos(JE) ≿ pos(NC) ∪ pos(JE) ≿ pos(NC) ⊆
pos(RC) ≿ pos(JE) ≿ (pos(NCO) ∪ pos(NC)) ∪ pos(JE) ≿ (pos(NCO) ∪ pos(NC))

We do this by showing

1. pos(JE) ≿ pos(NC)⊆ pos(JE) ≿ (pos(NCO) ∪ pos(NC)) and
2. pos(RC) ≿ pos(JE) ≿ pos(NC) ⊆ pos(RC) ≿ pos(JE) ≿ (pos(NCO) ∪ pos(NC)).

Condition 1 holds because pos(NC)⊆ pos(NCO) ∪ pos(NC) and the left hand set pos(JE) is identical on both sides of the ⊆ symbol. From this it also follows that condition 2 holds.

DISCLAIMER 29/11-2005:

The above argument is not correct since it ignores the fact that an assert operator is present in the BD_NotifyCustomersOptLoc sequence diagram. This assert operator gives rise to negative traces, so it is not true that neg(BlindDate3)=$\emptyset$ - so we have indeed added negative behavior. Therefore it would be sufficient to show that

$$\begin{aligned}
&\mathrm{pos}(\mathrm{RC}) \succsim \mathrm{pos}(\mathrm{JE}) \succsim \mathrm{pos}(\mathrm{NC}) \cup \mathrm{pos}(\mathrm{JE}) \succsim \mathrm{pos}(\mathrm{NC}) \subseteq \\
&\qquad (\mathrm{pos}(\mathrm{RC}) \succsim \mathrm{pos}(\mathrm{JE}) \succsim (\mathrm{pos}(\mathrm{NCO}) \cup \mathrm{pos}(\mathrm{NC})) \cup \\
&\qquad \mathrm{pos}(\mathrm{JE}) \succsim (\mathrm{pos}(\mathrm{NCO}) \cup \mathrm{pos}(\mathrm{NC}))) \cup \\
&\qquad \mathrm{neg}(\mathrm{BlindDate3}).
\end{aligned}$$

This follows trivially from

$$\begin{aligned}
&\mathrm{pos}(\mathrm{RC}) \succsim \mathrm{pos}(\mathrm{JE}) \succsim \mathrm{pos}(\mathrm{NC}) \cup \mathrm{pos}(\mathrm{JE}) \succsim \mathrm{pos}(\mathrm{NC}) \subseteq \\
&\qquad \mathrm{pos}(\mathrm{RC}) \succsim \mathrm{pos}(\mathrm{JE}) \succsim (\mathrm{pos}(\mathrm{NCO}) \cup \mathrm{pos}(\mathrm{NC})) \cup \\
&\qquad \mathrm{pos}(\mathrm{JE}) \succsim (\mathrm{pos}(\mathrm{NCO}) \cup \mathrm{pos}(\mathrm{NC})).
\end{aligned}$$

Therefore the conclusion still holds; BlindDate3 is a refinement of BlindDate1.