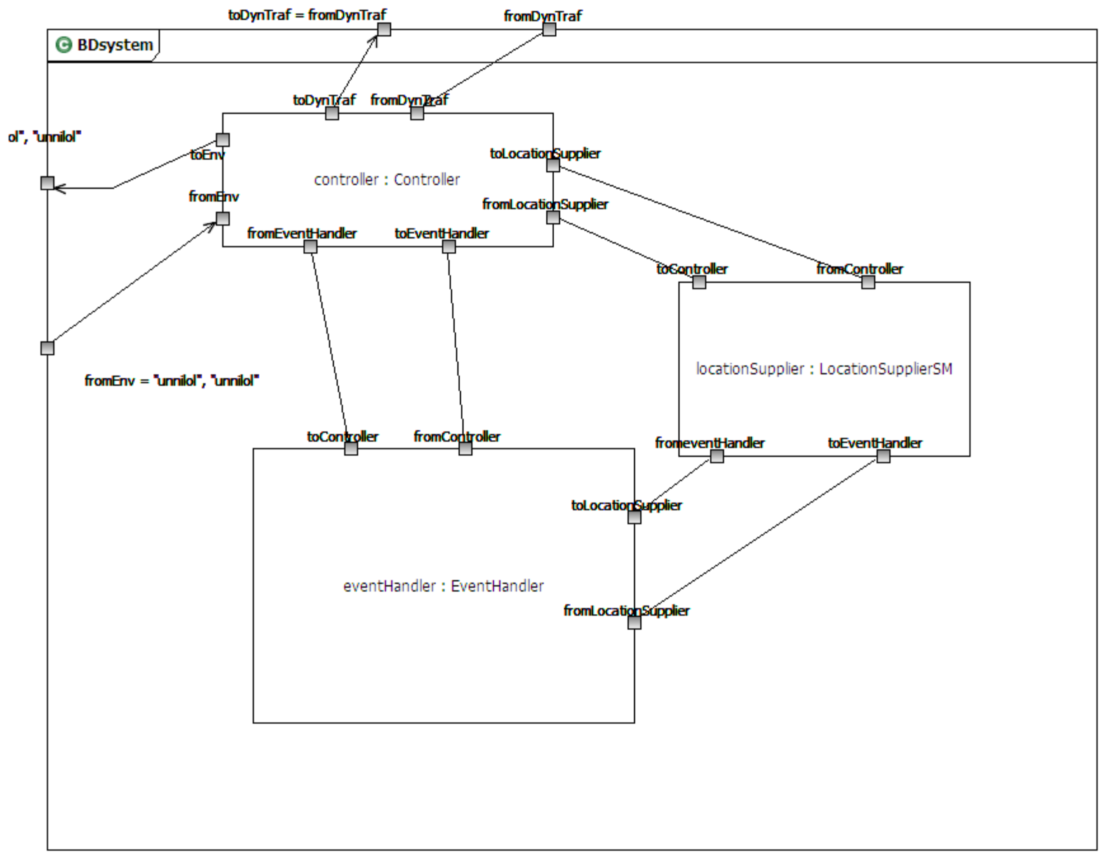
Description of Model

BDSystem

We will describe the major components of the system in turn. We begin with an overview of the entire system.

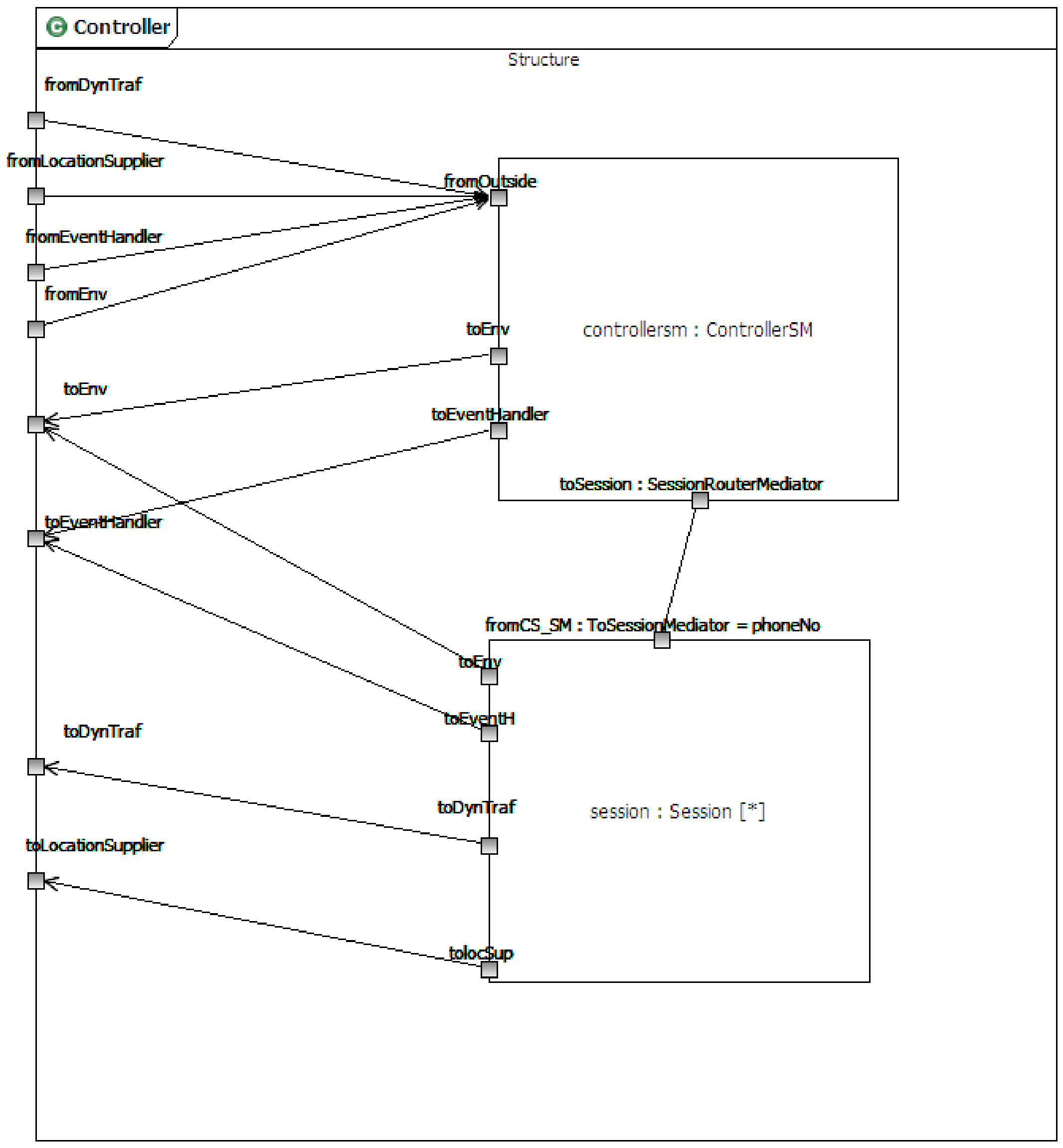
CS BDSystem



Figur 1 shows the composite structure of the blind date system. We have not made any changes to the specification. The major components are the Controller, the EventHandler and the LocationSupplier.

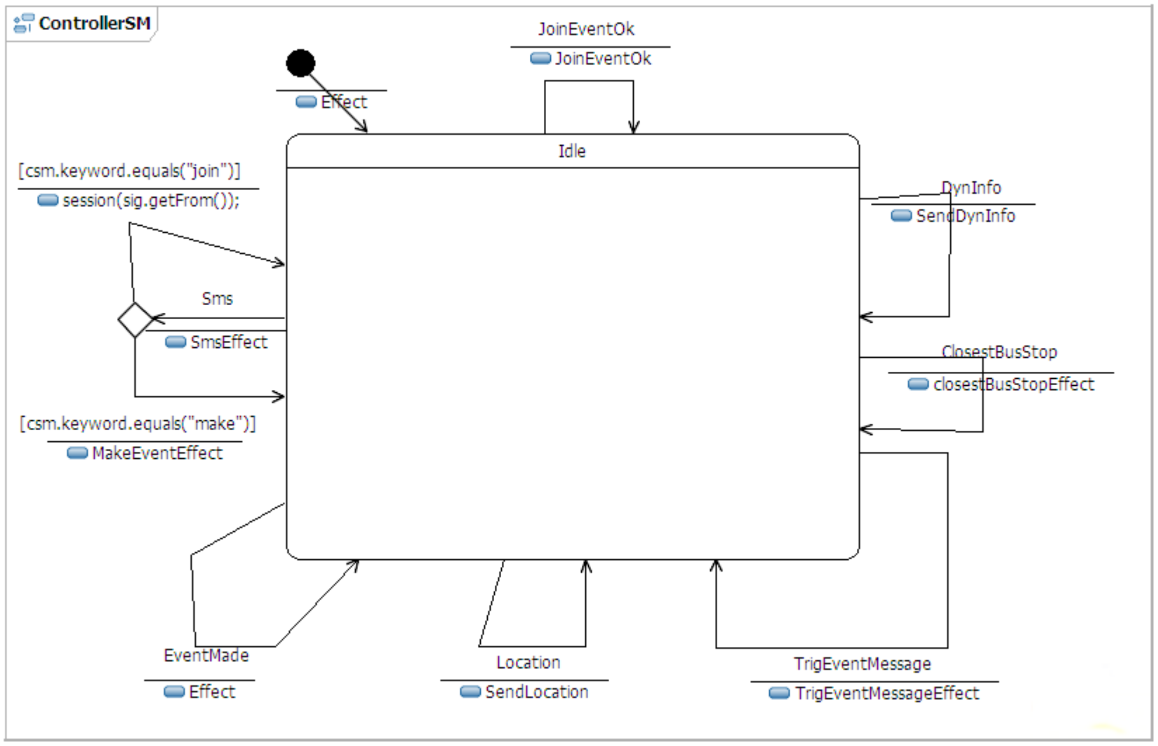
Controller

CS Controller



Figur 2 shows the composite structure of the Controller is essentially identical to the specification. It consists of one controller statemachine, called controllersm, and 0..* state machines for the active sessions. We have detailed that the port from controllersm to session is of type SessionRouterMediator and that the receiving port is of type ToSessionMediator. The SessionRouterMediator ensures that the correct session, i.e. the session with a specified phone number, receives signals.

SM Controller



Figur 3

The Controller operates as a traffic director – it receives messages from various sources and dispatches them to the correct recipient. We'll discuss each message in turn.

Initial effect. The controller does nothing on startup.

Sms. The system understands two types of SMS messages, join requests to join an existing event, and make requests to create new events. These messages have the form:

> Join: "join" <eventType> <time>
> Make: "make" <eventType> <time> <locName>

Each value must consist of exactly one word. Any other requests are ignored.

SmsEffect. When an SMS message is received, the controller parses the message and decides whether it is a join message or a make message, and sets the state machine attributes `keyword`, `eventType`, `time` and optionally `locName` accordingly.

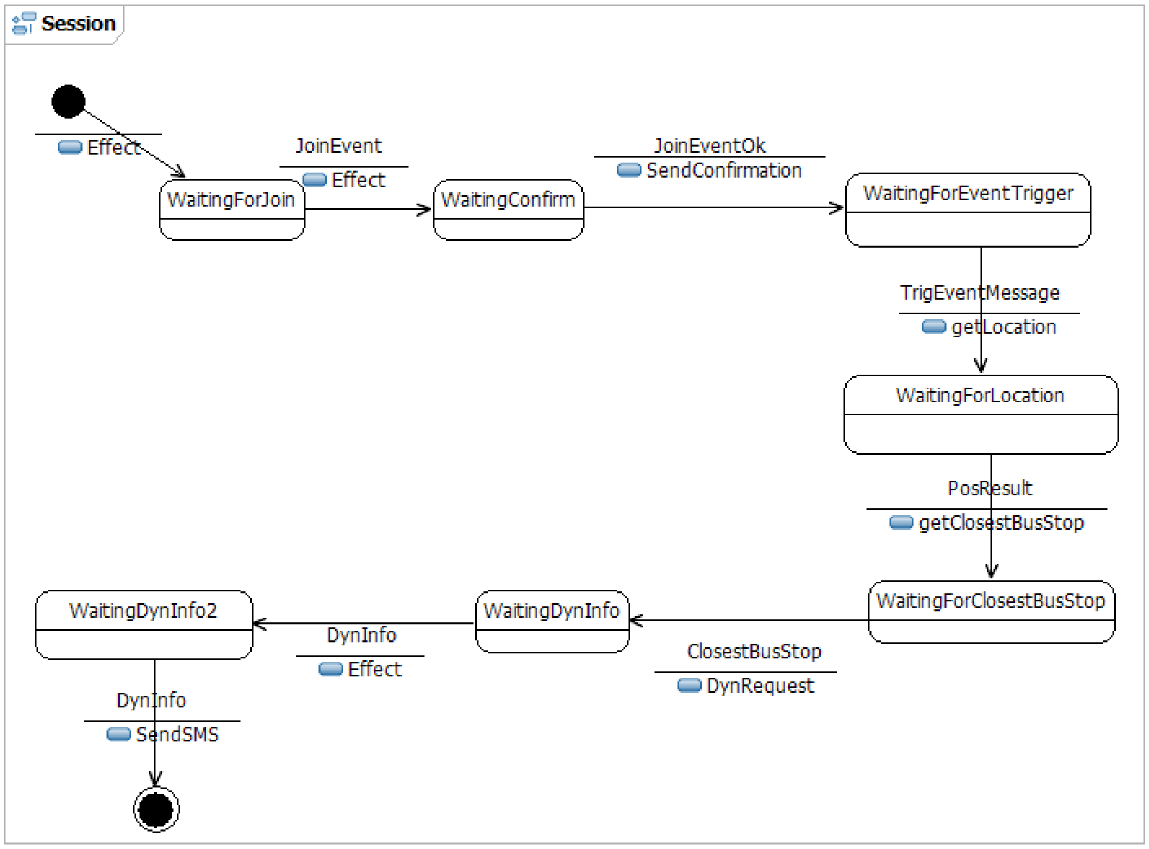
JoinEffect. If the message is a join request, a new session is created and associated with the phone number of the sender.

MakeEventEffect. Requests to create events are forwarded to the EventHandler as a MakeEvent message.

JoinEventOK, DynInfo, ClosestBusStop, TrigEventMessage, Location. All these signals are forwarded to the responsible session.

EventMade. Sends a confirmation SMS message to the person who initiated the creation request.

SM Session



Figur 4

A session essentially models a linear sequence of events and the states denote which event we are waiting for.

WaitingForJoin. Once a session has been created, it waits for information about the join request that created the session.

JoinEvent. When this information is obtained, the type and time of the event is recorded in the session attributes `eventType` and `time`, a join request is dispatched to the EventHandler for confirmation.

JoinEventOK. When the eventhandler confirms, a confirmation message is sent to user. The session then awaits the time when the event is supposed to begin.

TrigEventMessage. The session receives this signal a short time before the event begins. It responds by requesting the geographical location of the user's mobile phone.

PosResult. The PosResult signal contains information about the user's position. The session then sends a message to the LocationSupplier to find the nearest bus stop to the user.

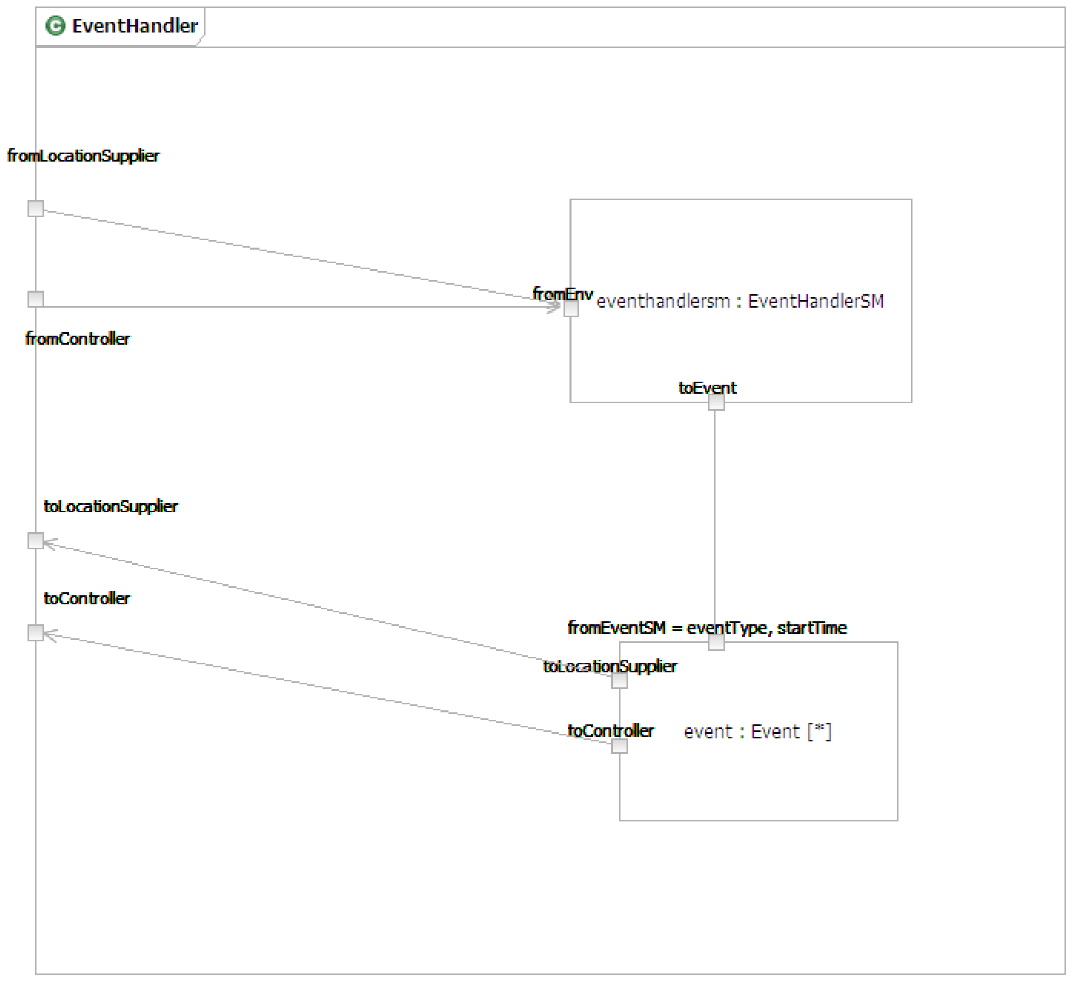
ClosestBusStop. After the LocationSupplier has determined the best bus stop, the session sends a DynRequest message to Trafikanten to find the bus schedules for the bus stop near the user.

DynInfo (1). When Trafikanten answers the query, we send a new request, this time asking for bus schedules for the bus stop near the event.

DynInfo (2). Finally, when Trafikanten answers the second request, a route is selected that will transport the participant to the event location from his current location. The session sends a notification SMS message to the user containing the eventType, event location, departure bus stop name, depature time and destination bus stop name.

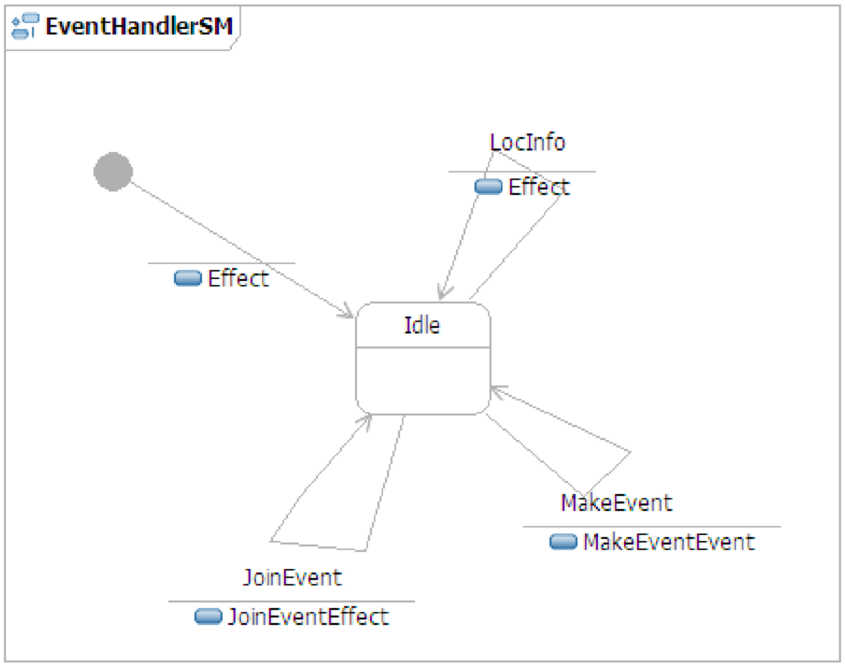
EventHandler

CS EventHandler



Figur 5
The composite structure of the EventHandler is identical to the specification. It consists of a main state machine, eventhandlersm, and one event state machine for each event. We have added the detail that each event is uniquely determined by its type and start time and supplied appropriate router mediators between the EventHandler and the Event.

SM EventHandler



Figur 6

The eventhandler understands the following messages, which may arrive at any time and in any order (except for the initial transition).

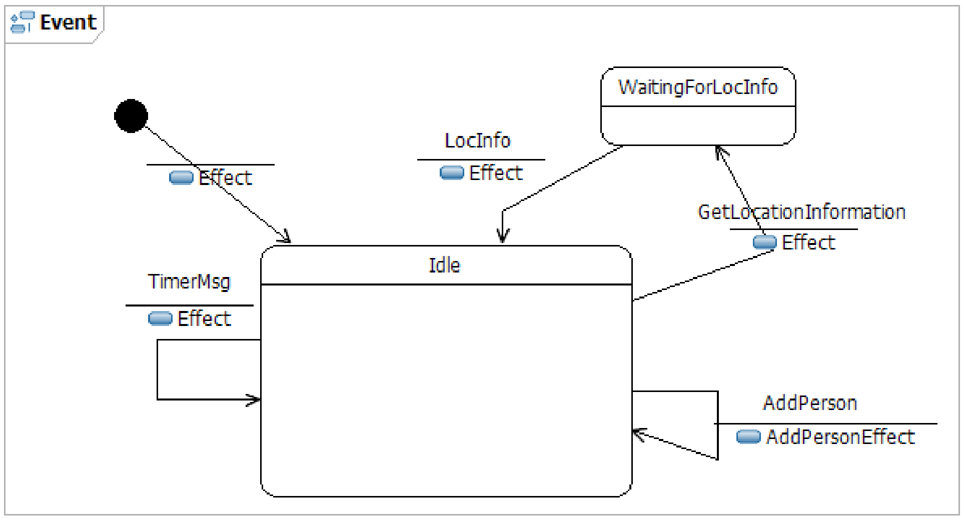
Initial. When created, the EventHandler creates a few events, just for testing purposes.

LocInfo. This message is forwarded to the correct event.

MakeEvent. Creates a new event statemachine and then sends a GetLocationInformation message to the event.

JoinEvent. Sends an AddPerson message to the event.

SM Event



Figur 7

The event responds to the following messages:

Initial. When the event is created, it sets a timer that will fire when the event is about to start. In our implementation, events always begin just a short time after a participant has joined the event. Then the timer expires, a TimerMsg will be dispatched to the event.

GetLocationInformation. This message is sent to the event by the EventHandler immediately after the event's creation. In response, the event sends a GetLocInfo message to the LocationSupplier. Until the LocationSupplier responds, the event will not be available.

LocInfo. When the response arrives, the Event registers the closest bus stop id and name in its attributes destBusStopID and destBusStopName. Afterwards, it sends an EventMade signal to the controller. The event is now ready to receive participants.

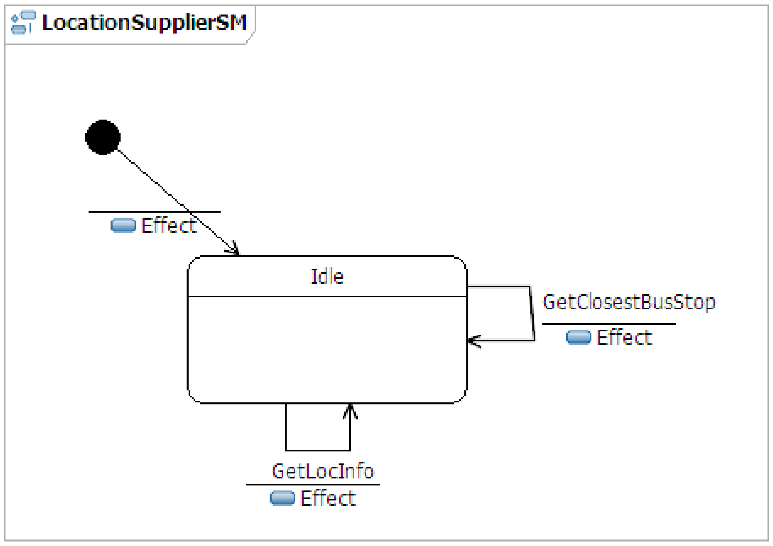
AddPerson. Adds the person, identified with his or her phone number, to the collection of participants for this event. Then sends a JoinEventOK message to the controller.

TrigEvent. Sends a TrigEventMessage to the controller for each participant.

LocationSupplier

The LocationSupplier, as described in the specification, contains knowledge of every bus stop and every possible event location. We have implemented this as `busStopList` and an `eventLocationList` attributes on the corresponding state machine.

SM LocationSupplier



Figur 8

The LocationSupplier sits around waiting for location questions to answer.

Initial. When created, the LocationSupplier loads the list of known bus stops from the file `37inf5150_ver2.txt`, which must be in the current directory. Using the information in this file, it populates the contents of the `busStopList` and `eventLocationList` attributes.

GetLocInfo. This message is sent by events when they are created. In response, the LocationSupplier sends a LocInfo message to the EventHandler containing the bus stop information for the created event.

GetClosestBusStop. When this message arrives, LocationSupplier tests all known bus stops to determine which one is closest to the given geographical position, using the GreatCircleDistance algorithm. When one bus stop has been selected, a ClosestBusStop signal is sent to the controller, containing the busStopId and busStopName of the selected bus stop.

Refinement

We have implemented a variant of the BlindDate system. In short, we have not implemented RegisterCustomer, but we have implemented the additional use case MakeEvent, which was described in the specification for BlindDate2. The remainder of this section will examine each sequence diagram and show that the implementation can be understood as a refinement of the original specification.

Generally, we do not handle many alternate sequences in our solution, but neither does the specification explicitly treat alternate situations in any detail.

Diagrams: SD BlindDate1, BD_BlindDate1

These specification diagrams refer to three lower-level diagrams: RegisterCustomer, JoinEvent and NotifyCustomers. RegisterCustomer is denoted optional. Our implementation does not implement RegisterCustomer, i.e. we have moved all traces that include this diagram to the set of negative traces. In refinement terminology, this is an instance of narrowing.

For this reason we do not comment on the sequence diagrams related to the register customer usecase.

Diagrams: SD JoinEvent, BD_JoinEvent, BD_Controller_JoinEvent

The specifications in these diagrams are implemented in full.

Diagram: SD NotifyCustomers, BD_NotifyCustomers, BD_Controller_NCust

Our solution implements the specification fully, but instead of the names “GetLocation” and “Location” the messages are called PosRequest and PosResult in our implementation. This is merely a change of name – semantically the traces are identical.

The specification is not very clear on the communication between the EventHandler and the Event state machines. Their lifelines are overlapping in the original diagrams. Here, our implementation can be understood as a detailing of the specification, in the sense that we have detailed the single, combined lifeline for EventHandler and Event in the original specification into two separate lifelines in the implementation. Detailing a lifeline is a refinement operation.

As a limitation, we have chosen to let an event begin a fixed time after it is created. It was shown in drop 1 that this can be considered as a refinement of the original specification.

Diagrams: SD MakeEvent, BD_MakeEvent, BD_EventHandler_MakeEvent

The fact that the Make Event usecase is a refinement of the original specification was established in the proposed solution to drop 1, and we do not repeat the argument here, but we will compare our implementation with the specification of MakeEvent given in BlindDate2.

Our implementation is almost compliant with the specification in these diagrams. The difference is as follows: in the specification sequence diagrams, events send the GetLocInfo message to the EventHandler after creation. In our implementation the Event does not send the GetLocInfo message until it receives a GetLocationInfo message from the EventHandler. However, due to the way the EventHandler creates events, this message is likely to arrive quickly after creation. Nevertheless, this difference implies that our implementation is not strictly a refinement of the specification, but due to time limitations we didn’t manage to change our model.

Except for this detail, our implementation is faithful to the specification.