6/7/07

Sensoria

016004
Software Engineering for Service-Oriented Overlay Computers

Automotive Case Study: LowOil Scenario

A SRML Model

Laura Bocchi[1] and Jose Bustos[1]

[1] Department of Computer Science, University of Leicester
University Road, Leicester LE1 7RH, UK
{bocchi,jab67}@mcs.le.ac.uk

Revision: Draft 1.2
Classification: PU

Contract Start Date: September 1, 2005 Duration: 48 months
Project Coordinator: LMU
Partners: LMU, UNITN, ULEICES, UWARSAW, DTU,
PISA, DSIUF, UNIBO, ISTI, FFCUL, UEDIN, ATX,
TILab, FAST, BUTE, S&N, LSS-Imperial, LSS-UCL

Integrated Project funded by the
European Community under the
"Information Society Technologies"
Programme (2002—2006)

1 Introduction

In this document, we present a SRML-P [1,2,3] model for the business process of *LowOil* described in the context of the Sensoria Automotive Case Study. The SRML model has been designed on the basis of the UML specification of the Automotive Case Study provided in [4]. Anyway, this document presents a new UML specification for compliance with the ongong work in Sensoria about UML externsions[1]. The *LowOil* agent addresses a low oil emergency within a vehicle by composing a number of internal components and invoking external services (e.g., for obtaining an accurate diagnosis of the problem and for booking a suitable on road repair service). The aim of this work is to gain experience from the modelling activity and to provide a feedback for the ongoing development of SRML. The *LowOil* case study provided a useful feedback on

- the notion of agent module (a module that does not provide any service and uses external services for achieving a private business goal),
- the notion of family of interaction events used to perform an arbitrary (i.e., determined at run time) number of interactions of the same type,
- some ideas on the relationship between UML structure diagrams and SRML module structure, and between UML activity diagrams and SRML business processes as we derived the SRML modules from a UML specification,
- the notion of message loss and unreliability.

Two SRML modules are proposed: *LowOil1* that assumes reliable messaging as provided by middleware and by the interacting services (this assumptions are expressed as SLA requirements) and *LowOil2* that copes with unreliability within the business logic.

Section 2 presents a summary of the case study, Section 3 defines the structure of the SRML modules, Section 4 discusses the request of reliability as a SLA, Section 5 comments on how unreliability is addressed in the busienss logic, Section 6 presents the conclusions. The appendixes present the SRML modules.

2 The Automotive Case Study: LowOil Scenario

We use structure diagrams for describing the structure of the module and activity diagrams to describe the orchestration. Section 2.1 presents the structure diagram of the low oil scenario, Section 2.2 specifies the orchestration in the simple scenario that assumes reliable communications. Section 2.2 specifies a number of error scenarios to address unreliable communications.

[1] http://www.pst.informatik.uni-muenchen.de:8080/Sensoria/T1.4

2.1 The Structure of the Service

The service includes the following components:

- *Communication System* orchestrates the interaction with the external services.
- *GPS* is a known component (i.e., not discovered) with a permanent connection with *Communication System*. It returns the current location of the vehicle.
- *External Diagnostic Service* is a known component that has a temporary connection with *Communication System*. It represents an external service that is bound at run time and is used just if/when necessary. It provides a diagnosis of the problem presented by the vehicle.
- *On Road Repair* must be discovered before the connection. It performs the booking of garage.

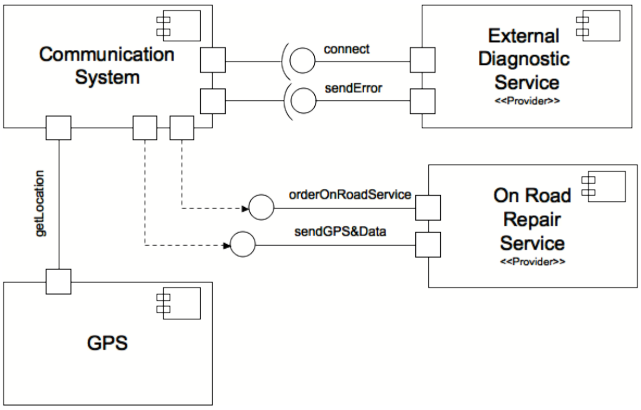


Figure 1: structure diagram of the Low Oil Level scenario

2.2 The Base Scenario - Assuming Reliabile Communication

The process of the basic low oil scenario is the following:

- The first activity belongs to the *Communication System* and is triggered by the notification of a low oil emergency.
- The vehicle, through the *Communication System,* establishes a connection with an *External Diagnostic Service* to have a more accurate diagnosis.
- In the mainwhile the service asks the current location of the vehicle to the *GPS* component, with a synchronous interaction; we are supposing the GPS position is always available.

- The vehicle, through the *Communication System,* communicates with an *On Road Repair Service* to obtain an appointment for repairing the vehicle. In this communication the location of the car is also communicated to the *On Road Repair Service*.

The activity diagram is presented in Figure 1.

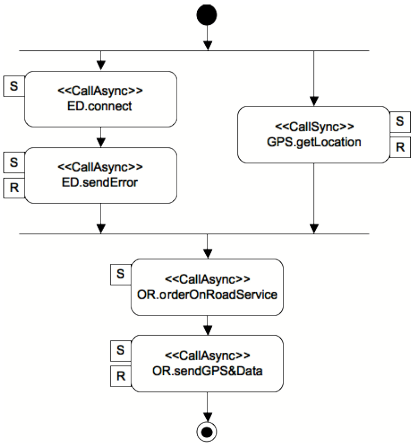


Figure 2: activity diagram of the Low Oil Level scenario (assuming reliable communication)

2.3 The Extended Scenario – Not Assuming Reliable Communication

In the first error scenario, illustrated in Figure 3, the communication *connect* is associated to a timer: if the participant does not synchronize on time, we assume that there was an error in the communication and we make another attempt with the same *External Diagnostic Service.* This protocol is repeated until a reply is received. Notice that in a more complex scenario we could communicate with another instance of *External Diagnostic Service*. In this document we keep trying the connection with the same service instance.

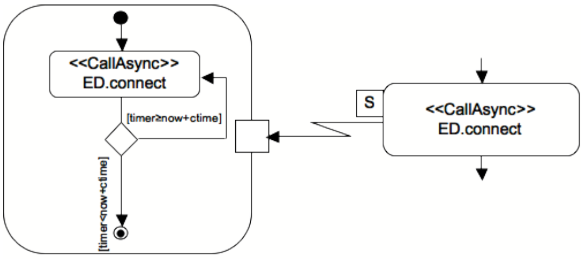


Figure 3: first error scenario: message loss of *connect*

The second error scenario, illustrated in Figure 4**Figure 4**, copes with message loss in the asynchronous communication *sendErrorData* with *ExternalDiagnosticService*. If a reply to the message is not received before a deadline, the whole protocol from the synchronous connection *connect* is repeated.

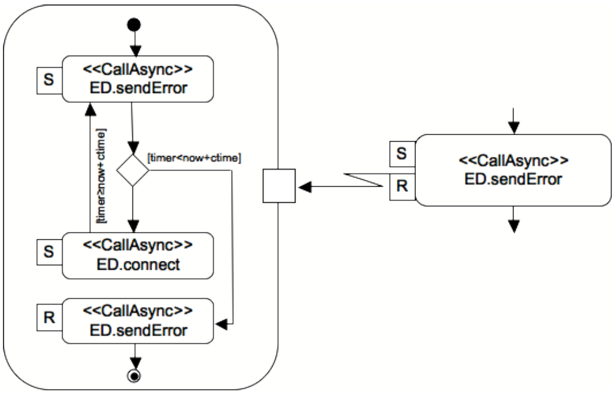


Figure 4: sequence diagram of the second error scenario: message loss of *sendErrorData*

The third error scenario is illustrated in Figure 5. If the functionality for retrieving the GPS position of the car is not available, the *Driver* is notified and instructed about the alternative procedure with a synchronous message.

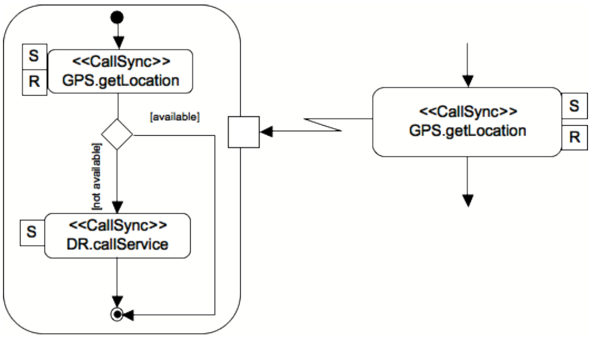


Figure 5: sequence diagram of the third error scenario: GPS not available

The fourth error scenario is illustrated in Figure 6. *Communication System* attempts a communication with *On Road Repair Service* to obtain an appointment. If no reply is received within an interval of time, a synchronous message is sent to ask the *Driver* to arrange the appointment manually, by phone.

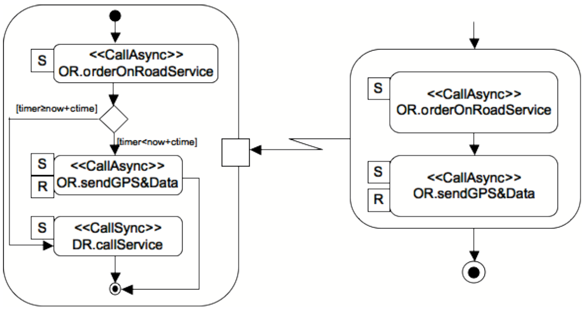


Figure 6: sequence diagram of the fourth error scenario: Onroad Repair service not available

3 The Structure of the Module

The structure of the module derives from the UML structure diagram. The components of the structure diagrams are modelled as nodes in the SRML module. In particular:

- *Communicating System* is the main orchestrator of the module and is represented as a component.
- The nodes with a permanent connection with *Communicating System* (i.e., *GPS*) are also represented as SRML service-components.
- The nodes with a temporary connection (either known or discovered) are represented as external interfaces. In our case all such components are providers, represented then as external-requires interfaces. A requester component with a temporary connection would be represented as an external provides interface.

We recall that SRML service-components are tightly coupled. External interfaces represent loosely coupled (typically) external services that are discovered/bound at run time. The UML components with a temporary connection but that are known a priori (e.g., *External Diagnostic Service*) are modelled as external interfaces whose *serviceID* is specified as a SLA requirement.

Figure 7 presents the structure of the SRML module for *lowOil1* (assuming reliability). *CommunicationSystem* and *GPS* are modelled as business roles. *ExternalDiagnosticService* and *OnRoadRepairService* are modelled as business protocols.

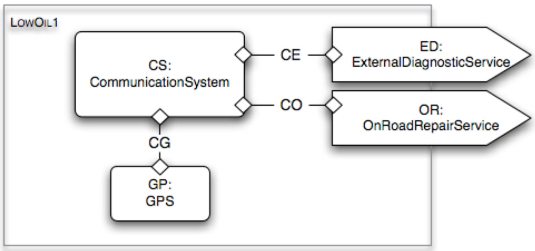


Figure 7: the structure of the SRML module LowOil1

The module does not provide any service and the process is started internally to the vehicle.

The module *LowOil2* (considering unreliabile communication) includes the additional component *DR* of type *Driver*. We chose to model *Driver* as a business role, and not as a EX-P or EX-R, since it is “statically bound” to the service and it is an internal element to the module car.

4 Requiring Reliability as a SLA

We modelled the request for reliable communication as a SLA in *LowOil1*. The constraints specifies an upper bound, represented by the constant *ctime*, for the replies of the interactions with external parties *Connect, onRoadService* and *SendErrorData*. The upper bound is defined in terms of the delay of the reply associated to an interaction (e.g., *connect.Reply*) and the delay of the wire (e.g., *CE.Delay*). The wire delay is considered twice because both the request and the response are subject to the delay. The constraints C_1 defines a specific service identifier of *ED*, as it represents a UML component that is known a priori but has temporary connection.

CONSTRAINT SYSTEM

S is <[0..1],max,min,0,1>
D is {n∈N:0≤n≤100}
V is {ED.ServiceID, ED.connect.Reply, ED.SendError.Reply,
OR.orderOnRoadService.Reply, CO.Delay, CE.Delay }

CONSTRAINTS

C_1 is <{ED.ServiceID},def$_1$> s.t. def$_1$(n)=1
if n=myExternalDiagnosticService;
C_2 is <{ED.connect.Reply,CE.Delay},def$_2$> s.t. def$_2$(n,m)=1
if n+2m<ctime and def$_2$(n,m)=0 otherwise;
C_3 is <{ED.sendError.Reply, CE.Delay},def$_3$> s.t. def$_3$(n,m)=1
if n+2m<ctime and def$_3$(n,m)=0 otherwise;
C_4 is <{OR.orderOnRoadService.Reply,CO.Delay},def$_4$> s.t.
def$_4$(n,m)=1 if n+2m<ctime and def$_4$(n,m)=0 otherwise;

5 Orchestration and Business Protocols

The orchestration of the business roles derive from the UML activity diagrams. We focus on the module *LowOil2* that presented a challenging scenario requiring to use families of interactions. We focus on the orchestration of *CommunicatingSystem*. Since every interaction event happens at most once during a session we use families of events denoted with different indexes. In this case study we use nested indexes: we have an arbitrary number of attempts of the interactions *connect* (i.e., first error scenario) and of the interaction *sendError* (i.e., second error scenario). The failure of *sendError* causes the reiteration of the process since the attempt of establishing a connection (i.e., *connect*). We use an asyncronous SRML interaction type to model the interactions with external parties. We use the notation *connect[i][j]* where *i* is the number of attempts to receive a reply to *sendError* and *j* is the number of attempts of performing the synchronous connection in the round *j*. The local variables *bl* (i.e., big loop represented by the first index of *connect*) and *sl* (i.e., small loop represented by the secong index of *connect*) keep record of the current state of this nested interaction. We use a local variable *timer* to set trigger the time. It follows the code fragment of the transition managing timeouts. The timeout can be related to: (1) the expiration of *connect*, (2) the expiration of *sendError* or (3) the expiration of *orderOnRoadService*. We use the local variable *phase* to keep track of the type of interaction currently associated with the timer.

```
transition Talert
 triggeredBy now=timer
 effects phase=connect ⊃ bl'=bl ∧ sl'=sl+1 ∧ s'=1
             ∧ timer'=now+ctime
    ∧ phase=sendError ⊃ bl'=bl+1 ∧ sl'=1 ∧ s'=1
             ∧ timer'=now+ctime ∧ phase'=connect
    ∧ phase=orderOnRoad ⊃ s'=5
             ∧ callService(sendError✉.diagnostic)
 sends  phase≠orderOnRoad  ⊃ connect[bl'][sl']🔔!
```

The business protocols of *LowOil2* require, for every message that can be lost, that eventually a message will be received and a synchronous connection attempt will be successful. We modelled the behaviour of the protocol using universal quantifiers.

$\forall_j(\neg\exists_{h<j}\mathtt{connect[i][h]}\text{🔔}?\ \mathbf{ensures}\ \exists_{k>j}\mathtt{connect[i][k]}\text{🔔}?)$

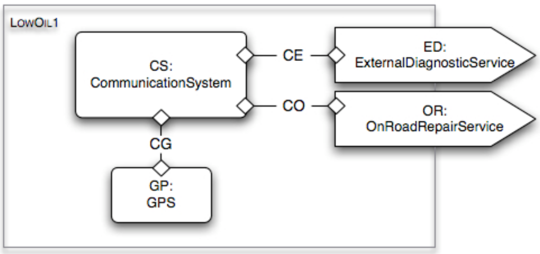
6 Concluding Remarks

7 References

1. J. L. Fiadeiro, A. Lopes, L. Bocchi (2006) *The SENSORIA Reference Modelling Language, Primitives for service description.* Sensoria Deliverable D1.1c.

2. J. Abreu, L. Bocchi, L. Fiadeiro, A. Lopes (2007) Specifying and composing interaction protocols for servic-oriented system modelling. In: J. Derrick, V. Jüri (eds) *Web Services and Formal Methods. LNCS, vol 4576*. Springer, Berlin Heidelberg New York, pp 358–373
3. J. L. Fiadeiro, A. Lopes, L. Bocchi (2006) A formal approach to service-oriented architecture. In: M. Bravetti, M. Nunez, G. Zavattaro (eds) *Web Services and Formal Methods. LNCS, vol 4184*. Springer, Berlin Heidelberg New York, pp 193–213
4. M. Banci, A. Fantechi, S. Giannini, F. Santanni (2006). Automotive Case Study: a UML Description Scenario. (Available at http://www.pst.informatik.uni-muenchen.de:8080/AutomotiveCaseStudy/Description+by+UML+Sequence+Diagrams+of+Scenarios)

Appendix 1 – LowOil assuming reliability



LOWOIL1 consists of:

- ED – requires-interface, of type *ExternalDiagnosticService*;
- OR – requires-interface, of type *OnRoadRepairService*;
- CS – component of type *CommunicationSystem*;
- GP – component of type *GPS*;
- *CE, CO, CG* – the internal wires.

MODULE LowOil1 **is**

DATATYPE

sorts: problemData, diagnosticData, location, location ∪ NULL, boolean, natural

COMPONENTS

GP: GPS
CS: CommunicationSystem

REQUIRES

ED: ExternalDiagnostic
OR: OnRoadRepairService

CONSTRAINT SYSTEM

S is <[0..1],max,min,0,1>

D is {n∈N:0≤n≤100}

V is {ED.ServiceID, ED.connect.Reply, ED.SendError.Reply, OR.orderOnRoadService.Reply, CO.Delay, CE.Delay}

CONSTRAINTS

C_1 is <{ED.ServiceID},def_1> s.t. def_1(n)=1 if n=myExternalDiagnosticService;

C_2 is <{ED.connect.Reply,CE.Delay},def_2> s.t. def_2(n,m)=1 if n+2m<ctime and def_2(n,m)=0 otherwise;

C_3 is <{ED.sendError.Reply, CE.Delay},def_3> s.t. def_3(n,m)=1 if n+2m<ctime and def_3(n,m)=0 otherwise;

C_4 is <{OR.orderOnRoadService.Reply,CO.Delay},def_4> s.t.
def_4(n,m)=1 if n+2m<ctime and def_4(n,m)=0 otherwise;

WIRES

GP GPS		**CG**		**CS** Communication System
tll getLocation	S_1	**AskTllEmptyI** [location]	R_1	**ask** getLocation

CS Communication System		**CE**		**ED** ExternalDiagnostic
s&r connect	S_1	**Straight**	R_1	**r&s** connect
s&r sendError 🔔 problem ✉ diagnosis	S_1 i_1 o_1	**Straight** I[problemData] O[diagnosticData]	R_1 i_1 o_1	**r&s** sendError 🔔 problem ✉ diagnosis

CS Communication System		**CO**		**OR** OnRoadRepairService
s&r orderOnRoadService	S_1	**Straight**	R_1	**r&s** orderOnRoadService
s&r sendGPS&Data 🔔 vehicleLocation diagnosis	S_1 i_1 i_2	**Straight** I[location, diagnosticData]	R_1 i_1 i_2	**r&s** sendGPS&Data 🔔 vehicleLocation diagnosis

END MODULE

SPECIFICATIONS

BUSINESS ROLE GPS **is**

INTERACTIONS

tll getLocation():location ∪ NULL

ORCHESTRATION

local vehicleLocation()→location ∪ NULL
transition
  triggeredBy getLocation()
  send vehicleLocation()

END BUSINESS ROLE

```
BUSINESS ROLE CommunicationSystem is

  INTERACTIONS
    ask getLocation():location ∪ NULL
    s&r connect
    s&r sendError
      🔔  problem: problemData
      ✉  diagnosis: diagnosticData
    s&r orderOnRoadService
    s&r sendGPS&Data
      🔔  diagnosis: diagnosisData
          vehicleLocation: location

  ORCHESTRATION
    local s:[0..5], position:location ∪ NULL, get-
    Data()→problemData, once:boolean
    initialisation
      s=0 ∧ ¬once ∧ position=NULL
    termination
      s=5
    transition Init 1
     | triggeredBy true
     | guardedBy s=0
     | effects s'=1
     | sends connect🔔!
    transition Init 2
     | triggeredBy true
     | guardedBy s=0 ∧ ¬once
     | effects position'=getLocation() ∧ once'
    transition Connected
     | triggeredBy connect✉?
     | guardedBy s=1
     | effects s'=2
     | sends sendError🔔!
     |    ∧ sendError🔔.problem=getData()
    transition Join
     | triggeredBy sendError✉?
     | guardedBy s=2 ∧ position≠NULL
     | effects s'=3
     | sends orderOnRoadService🔔!
    transition SendData
     | triggeredBy orderOnRoadService✉?
     | guardedBy s=3
     | effects s'=4
     | sends sendGPS&Data🔔!
     |    ∧ sendGPS&Data🔔.diagnosis=sendError✉.diagnosis
     |    ∧ sendGPS&Data🔔.vehicleLocation=position
    transition Confirmation
     | triggeredBy sendGPS&Data✉?
     | guardedBy s=4
     | effects s'=5
                                                    END BUSINESS ROLE

BUSINESS PROTOCOL ExternalDiagnosticService is
```

```
   INTERACTIONS
      r&s connect
      r&s sendError
         🔔 problem:problemData
         ✉ diagnosis: diagnosticData
   BEHAVIOUR
      initiallyEnabled connect🔔?
      connect✉! enables sendError🔔?
                                          END BUSINESS PROTOCOL

BUSINESS PROTOCOL OnRoadRepairService is
   INTERACTIONS
      r&s orderOnRoadService
      r&s sendGPS&Data
         🔔 vehicleLocation:location
         ✉ diagnosis:diagnosticData
   BEHAVIOUR
      initiallyEnabled orderOnRoadService🔔?
      orderOnRoadService✉! enables sendGPS&Data🔔?
                                    END BUSINESS PROTOCOL

INTERACTION PROTOCOL Straight  is
   ROLE A
      snd S1

   ROLE B
      rcv R1

   COORDINATION
         S1 ≡ R1

INTERACTION PROTOCOL Straight.I(d1)O(d2) is
   ROLE A
      r&s S1
         🔔 i1:d1
         ✉ o1:d2


   ROLE B
      s&r R1
         🔔 i1:d1
         ✉ o1:d2

   COORDINATION
         S1 ≡ R1
         S1.i1=R1.i1
         S1.o1=R1.o1



INTERACTION PROTOCOL Straight.I(d1,d2) is
   ROLE A
```

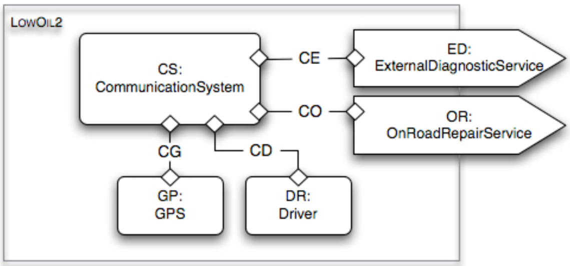
```
    r&s S_1
        🔔 i_1:d_1
           i_2:d_2

ROLE B
    s&r R_1
        🔔 i_1:d_1
           i_2:d_2

COORDINATION
        S_1 ≡ R_1
        S_1.i_1=R_1.i_1
        S_1.i_2=R_1.i_2
```

```
INTERACTION PROTOCOL AskTllEmptyI(d_1) is
  ROLE A
    ask S_1():d_1
  ROLE B
    tll R_1():d_1
  COORDINATION
    S_1() = R_1()
```

Appendix 2 – LowOil not assuming reliability



LowOil2 consists of:

- *ED* – requires-interface, of type *ExternalDiagnosticService*;
- *OR* – requires-interface, of type *OnRoadRepairService*;
- *DR*- component of type *Driver*;
- *GP*- component of type *GPS*;
- *CS* – component of type *CommunicationSystem*;
- *CG, CE, CD, CO* – the internal wires.

```
MODULE LowOil2 is

  DATATYPE
```

sorts: problemData, diagnosticData, location, location ∪ NULL ∪ NA, boolean, natural

COMPONENTS

CS: CommunicationSystem
DR: Driver
GP: GPS

REQUIRES

ED: ExternalDiagnostic
OR: OnRoadRepairService

WIRES

GP GPS	◇—	**CG**	—◇	**CS** Communication System
tll getLocation	S_1	**AskTllEmptyI** [location]	R_1	**ask** getLocation

DR Driver	◇—	**CD**	—◇	**CS** Communication System
ask callService	S_1	**AskTllEmptyO** [diagnosticData]	R_1	**rpl** callService
ask providePos	S_1	**AskTll**	R_1	**rpl** providePos

CS Communication System	◇—	**CE**	—◇	**ED** ExternalDiagnostic
s&r connect	S_1	**Straight**	R_1	**r&s** connect
s&r sendError 🔔 problem ✉ diagnosis	S_1 i_1 o_1	**Straight** I[problemData] O[diagnosticData]	R_1 i_1 o_1	**r&s** sendError 🔔 problem ✉ diagnosis

CS Communication System	◇—	**CO**	—◇	**OR** OnRoadRepairService
s&r orderOnRoadService	S_1	**Straight**	R_1	**r&s** orderOnRoadService
s&r sendGPS&Data 🔔 vehicleLocation diagnosis	S_1 i_1 i_2	**Straight** I[location, diagnosticData]	R_1 i_1 i_2	**r&s** sendGPS&Data 🔔 vehicleLocation diagnosis

```
                                                        END MODULE

SPECIFICATIONS

  BUSINESS ROLE GPS is

    INTERACTIONS
      tll getLocation():location ∪ NULL ∪ NA

    ORCHESTRATION
      local vehicleLocation()→location∪NULL∪NA

      transition
        | triggeredBy getLocation()
        | send vehicleLocation()
                                                 END BUSINESS ROLE

  BUSINESS ROLE Driver is

    INTERACTIONS
      ask providePos()
      ask callService(diagnosticData)
    ORCHESTRATION
      transition
        | triggeredBy providePos()
      transition
        | triggeredBy callService(d)
                                                 END BUSINESS ROLE

  BUSINESS ROLE CommunicationSystem is

    INTERACTIONS
      ask getLocation():location ∪ NULL ∪ NA
      s&r connect[i:natural][j:natural]
      s&r sendError[i:natural]
        🔔  problem: problemData
        ✉  diagnosis: diagnosticData
      s&r orderOnRoadService
      s&r sendGPS&Data
        🔔  diagnosis: diagnosticData
            vehicleLocation: location
      rpl providePos()
      rpl callService(diagnosticData)

    ORCHESTRATION
      local s:[0..6], position:location∪NULL∪NA, get-
      Data()→problemData, bl,sl:natural, timer:time,
      once:boolean, phase:{connect,sendError,orderOnRoad}
      initialisation
        s=0 ∧ position=NULL ∧ bl=sl=1 ∧ timer=∞ ∧ ¬once
      termination
        s=6

      transition Init1
        | triggeredBy true
        | guardedBy s=0
        | effects s’=1
```

```
     ∧ timer'=now+ctime
     ∧ phase'=connect
   sends connect[bl][sl]🔔!
transition Init2
   triggeredBy true
   guardedBy s=0 ∧ ¬once
   effects  position'=getLocation() ∧ once'
transition Connect
   triggeredBy connect[i][j]✉?
   guardedBy s=1 ∧ bl=i ∧ sl=j
   effects phase'=sendError
     ∧ timer'=now+ctime
   sends sendError[i]🔔!
     ∧ sendError.error=getData()
transition Talert
   triggeredBy now=timer
   guardedBy
   effects phase=connect ⊃ bl'=bl ∧ sl'=sl+1 ∧ s'=1
              ∧ timer'=now+ctime
     ∧ phase=sendError ⊃ bl'=bl+1 ∧ sl'=1 ∧ s'=1
              ∧ timer'=now+ctime ∧ phase'=connect
     ∧ phase=orderOnRoad ⊃ s'=5
              ∧ callService(sendError✉.diagnostic)
   sends  phase≠orderOnRoad  ⊃ connect[bl'][sl']🔔!
transition SendingError
   triggeredBy sendError[i]✉?
   guardedBy s=2 ∧ position ≠ NULL
   effects s'=3
     ∧ position=NA ⊃ providePosition()
     ∧ timer'=now+ctime
     ∧ phase'=orderOnRoad
   sends orderOnRoadService🔔!
transition RepairBooking
   triggeredBy orderOnRoadService✉?
   guardedBy s=3
   effects s'=4 ∧ timer'=∞
   sends sendGPS&Data🔔!
     ∧ sendGPS&Data.vehicleLocation=position
     ∧ sendGPS&Data.diagnosis=sendError✉.diagnosis
transition ConfirmDates
   triggeredBy sendGPS&Data✉?
   guardedBy s=4
   effects s'=5
                                              END BUSINESS ROLE

BUSINESS PROTOCOL ExternalDiagnosticService is
  INTERACTIONS
    r&s connect[i:natural][j:natural]
    r&s sendError[i:natural]
       🔔 problem:problemData
       ✉ diagnosis:diagnosticData
  BEHAVIOUR
    $\forall_j$(¬$\exists_{h<j}$connect[i][h]🔔?    ensures    $\exists_{k>j}$connect[i][k]🔔?)
    connect[i][j]✉! enables sendError[i]🔔?
    $\forall_i$(¬$\exists_{h<i}$sendError[h]🔔? ensures $\exists_{k>i}$sendError[k]🔔?)
```

```
                              END BUSINESS PROTOCOL
```

```
BUSINESS PROTOCOL OnRoadRepair is
   INTERACTIONS
      r&s orderOnRoadService
      r&s sendGPS&Data
         🔔 vehicleLocation:location
            diagnosis:diagnosticData
   BEHAVIOUR
      initiallyEnabled orderOnRoadService🔔?
      orderOnRoadService✉! enables sendGPS&Data🔔?
                              END BUSINESS PROTOCOL
```

```
INTERACTION PROTOCOL Straight  is
   ROLE A
      snd $S_1$

   ROLE B
      rcv $R_1$

   COORDINATION
         $S_1 \equiv R_1$
```

```
INTERACTION PROTOCOL Straight.I($d_1$)O($d_2$) is
   ROLE A
      r&s $S_1$
         🔔 $i_1$:$d_1$
         ✉ $o_1$:$d_2$

   ROLE B
      s&r $R_1$
         🔔 $i_1$:$d_1$
         ✉ $o_1$:$d_2$

   COORDINATION
         $S_1 \equiv R_1$
         $S_1.i_1=R_1.i_1$
         $S_1.o_1=R_1.o_1$
```

```
INTERACTION PROTOCOL Straight.I($d_1$,$d_2$) is
   ROLE A
      r&s $S_1$
         🔔 $i_1$:$d_1$
            $i_2$:$d_2$

   ROLE B
      s&r $R_1$
         🔔 $i_1$:$d_1$
```

```
            $i_2$:$d_2$

   COORDINATION
         $S_1 \equiv R_1$
         $S_1.i_1$=$R_1.i_1$
         $S_1.i_2$=$R_1.i_2$

INTERACTION PROTOCOL AskTllEmptyI($d_1$) is
   ROLE A
      ask $S_1$():$d_1$
   ROLE B
      tll $R_1$():$d_1$
   COORDINATION
      $S_1$() = $R_1$()

INTERACTION PROTOCOL AskTllEmptyO($d_1$) is
   ROLE A
      ask $S_1$($d_1$)
   ROLE B
      tll $R_1$($d_1$)
   COORDINATION
      $S_1$(x) = $R_1$(x)

INTERACTION PROTOCOL AskTll is
   ROLE A
      ask $S_1$()
   ROLE B
      tll $R_1$()
   COORDINATION
      $S_1$() = $R_1$()
```