



Sensoria

016004

*Software Engineering for Service-Oriented
Overlay Computers*

Automotive Scenario: Illustrating Service Specification

FAST Report # 2, 2007

Author(s): Dominik Berndl (FAST), Nora Koch (LMU & FAST)
Date: August 31, 2007

Revision: Draft
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, MIP, ATXT



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

Executive Summary

The document reports the use of scenarios of the automotive case study in the specification of services. Scenarios are used to illustrate different specification techniques developed or adapted in the SENSORIA project for the engineering of service-oriented systems.

This is a DRAFT version of the report which goal is to provide a detail specification of a selected set of scenarios of the automotive case study. The current draft version focus on the UML specification of the scenarios. It will be successively improved with other specifications, such as those based on SRML and CoWS. Analysis techniques applied to these scenarios will be included as well. The selected are the on road car repair and accident scenarios as they resulted in the more interesting scenarios from the orchestration and analysis point of view.

Contents

1	Introduction	3
2	Automotive Service-Oriented Architecture	3
3	Specification of the On Road Car Repair Scenario	5
3.1	Modelling Structural Aspects of Services in Detail.....	6
3.2	Modelling Behavioural Aspects.....	8
3.2.1	Modelling Orchestration	8
3.2.2	Modelling Compensation	9
3.2.3	Modelling the Communication Flow	11
3.3	Quantitative and Qualitative Analysis	15
3.3.1	Model Checking Orchestration	15
4	Specification of the Airbag Scenario.....	18
4.1	Modelling the scenario.....	18
4.2	System overview.....	19
4.3	Workflow in the airbag scenario.....	20
4.4	Activities of the communication system.....	20
5	Specification of the Reconfigurable Route Planning Scenario	22
6	Specification of the Road Sights Scenario	24
7	Specification of the Dinner Break Scenario	25
8	Bibliography	25

1 Introduction

One main element of SENSORIA are realistic case studies for different important application areas including e-business, telecommunications and automotive to provide a context for developing intuitions that can feed and challenge the research process according to the expectations of society and its economy, discussing and communicating ideas among partners-and finally communicating research results to and getting feedback from the research community at large, both in industry and academia.

Most of the case studies are defined by the industrial SENSORIA partners as to provide continuous practical challenges for the new techniques of Services Engineering and to serve for demonstrating the research results. This report concentrates on scenarios in the automotive domain.

Section 2 gives an overview of the Automotive Service-Oriented Architecture. In the following sections the use of scenarios of the automotive case study in the specification of services is reported. Different specification techniques developed or adapted to the SENSORIA project using the automotive scenario will introduced.

2 Automotive Service-Oriented Architecture

Figure 1 provides an overview of the Service-Oriented Architecture (SOA) of a vehicle. The architectural modules are represented as nodes in an UML deployment diagram. This type of diagram has been selected to show the distribution of the components within the different platforms and devices of the vehicle, and those belonging to the vehicle environment. The focus is also to model the different type of communications between the vehicle and its environment.

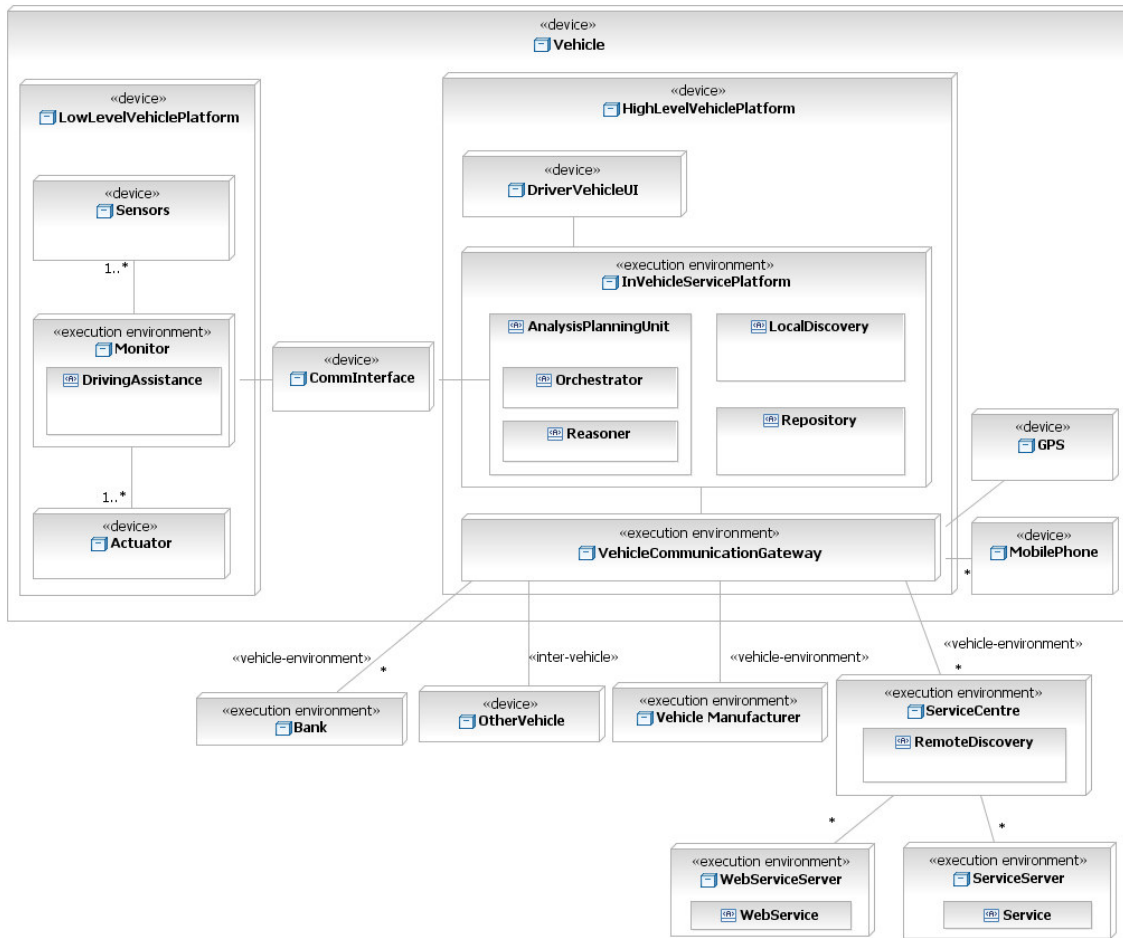


Figure 1: Automotive SOA

Our automotive service-oriented architecture consists of the following modules:

Node	Description
Vehicle	Represents the physical entity that is mean to carry or transport something. In the context of SENSORIA a vehicle contains sensors and is able to determine its geographical position and to interact with the external world. It is modelled as a device node.
Hardware/Low Level Vehicle Platform	Acts as a container for both the hardware and the low-level software components of the vehicle. Critical driving assistance services as ABS or ESP and their related sensor software systems are deployed to this low architectural level to ensure minimal response time.
Sensor	Contains all relevant sensors necessary to observe the vehicle’s status. Each sensor has to provide information to the <i>Monitor</i> .
Monitor	Manages and monitors all vehicle sensors, alerts in case of a sensor indicating an event. Very critical situations (e.g. wheel spin) detected are reported to the <i>Actuator Software</i> while other sensor indications (e.g. low fuel or oil level) are submitted to the <i>Analysis and Service Planning</i> node.
Driving assistance	Contains all low-level driving assistance implemented functionalities (e.g. ABS), which are triggered by the <i>Actuators</i> and deployed on the <i>Monitor</i> .
Actuator	Triggers fully-automatically the on-vehicle (low-level) driving assistance systems like ABS, anti slipping assistance and stability assistance. Note that no complex diagnostics

	and planning have to be performed by the <i>Actuator</i> .
Communication Interface	Enables communication between low- and high-level platforms of the vehicle.
High Level Vehicle Platform	A computing platform to perform the higher computational functionalities of the vehicle.
Driver/Vehicle UI	Communication interface between driver and vehicle. The driver receives information from the active services and can enter commands to trigger, stop or customise them.
Analysis and Planning Unit	Analyses events reported by the <i>Monitor</i> or <i>Driver/Vehicle UI</i> or the <i>Vehicle Communication Gateway</i> and plan the corresponding actions while typically human interaction and/or complex communication patterns are involved. The <i>Internal Service Repository</i> is asked for an appropriate service for specific actions. An example is the lookup of an onboard diagnostic service when the <i>Monitor</i> reports abnormal situations. It must provide mechanisms for lookup (of the service repository), discovery and registration of services.
Orchestrator	Architectural element that is in charge to achieve a goal by mean of a composition of services.
Reasoner	Analyses and selects services based on established criteria.
Driver/Vehicle UI	Physical input/output device that allows for communication between driver and vehicle. The driver receives information from the active services and can enter commands to trigger, stop or customise them.
Local Discovery	Local service for finding appropriate services.
Repository	Internal repository of services, where in-car services or software needed for consuming external services are locally stored
Vehicle Communication Gateway	Abstracts sending of a message to an external component (e.g. another vehicle or a server) from the underlying communication technology and protocol. The best suitable communication technique (e.g. UMTS, GPRS, WLAN) is selected dynamically and transparently to the sender of the message.
Mobile Phone	Device used to build up GPRS or UMTS connections. The mobile phone can be built-in the vehicle or provided by the driver.
GPS	Receiver for Global Positioning System information.
Bank	Commercial or state institution that provides financial services.
Vehicle Manufacturer	Manufacturer of the vehicle. In the SENSORIA context it also provides services, such as remote diagnostic and car repair.
Service Centre	Remote execution platform offering discovery of services and applications supporting the service requests.
Remote Discovery	Remote service for finding appropriate services.
Service Server/Provider	Remote execution platform offering services.
Web Service Server	Remote execution platform containing services available on the Web.
Service	An abstract resource that represents a capability of performing tasks that form a coherent functionality from the point of view of providers entities and requesters entities (SENSORIA Glossary).
Web Service	Service that is available on the Web.

Table 1: Elements of SENSORIA automotive architecture

3 Specification of the On Road Car Repair Scenario

In the car repair scenario, the diagnostic system reports a severe failure in the car engine, for example, the vehicles oil lamp reports low oil levels. This triggers the in-vehicle diagnostic system to perform an analysis of the sensor values. The diagnostic system reports a problem with the pressure in one cylinder head, and that the car is no longer driveable, and sends a message with the diagnostic data as well as the vehicle's GPS data to the repair server. Based on availability and the driver's preferences, the service discovery system identifies an

adequate repair shop in the area, whose phone number and address are displayed on the in-vehicle display. The driver then makes an appointment with the repair shop; the diagnostic data is automatically transferred to the repair shop, which is then able to identify the parts needed to perform the repair.

After the appointment with the shop has been completed, the service discovery system identifies a towing service, providing to both the GPS data of the stranded vehicle. The driver makes an appointment with the towing service, and the vehicle is towed to the shop. The selection of services takes into account personalised policies and preferences of the driver. We assume that the owner of the car has to deposit a security payment before being able to order services.

This scenario belongs to the category “car repair assistance”.

The car repair scenario is a widely used scenario for demonstrating techniques developed in the context of Sensoria. Wirsing et al. present a summary of main techniques for the semantic-based development of service-oriented systems arising in the context of the Sensoria project. The different techniques introduced in the paper include service-oriented extensions to the UML, a mathematical basis formed by a family of process calculi, a language for expressing context-dependent soft constraints and preferences, qualitative and quantitative analysis methods, and model transformations from UML to process calculi.

3.1 Modelling Structural Aspects of Services in Detail

Service oriented architectures are highly dynamic because services are only loosely coupled, i.e., a service often needs to be discovered before it is connected and can be disconnected at run-time as well. Hence, different modelling features are required to express evolving connections. In addition to UML deployment diagrams, which give a static view of the architecture, a representation showing the evolution of architecture is required. Baresi, Heckel, Thöne and Varró propose the construction of models visualising the functional aspects encapsulated in business-related components [BH+2006]. These UML structure diagrams are used to represent the evolving connections within the service-oriented architecture of the vehicle and its environment. Figure 2 shows the car internal components, a permanent connection to the local discovery service of the car and a temporary connection to the location service delivering GPS data. Remote services such as tow truck and garage and car rental are offered by the component called OnRoadService. The OnRoadService provides interfaces, where the offered services can be accessed. After publishing, the local discovery service knows the description of the published services, so that it can pass these descriptions to service requesters at service discovery time.

Three different types of connections are identified: discovery connection, permanent connection (as in modelling of non service oriented architectures) and temporary connections. For more details about the last two types the reader is referred to [3]. The discovery connection is based on the information provided by a discovery service. We can observe these three types of connections in the service oriented vehicle architecture:

For a permanent service we select a UML connector without interfaces as shown between the Local Discovery and the Vehicle Communication System in Figure 2. A temporary connection from the car on-board system to the car manufacturer discovery service is graphically represented by a UML connector with interfaces. Figure 3 shows some of the interfaces of the components.

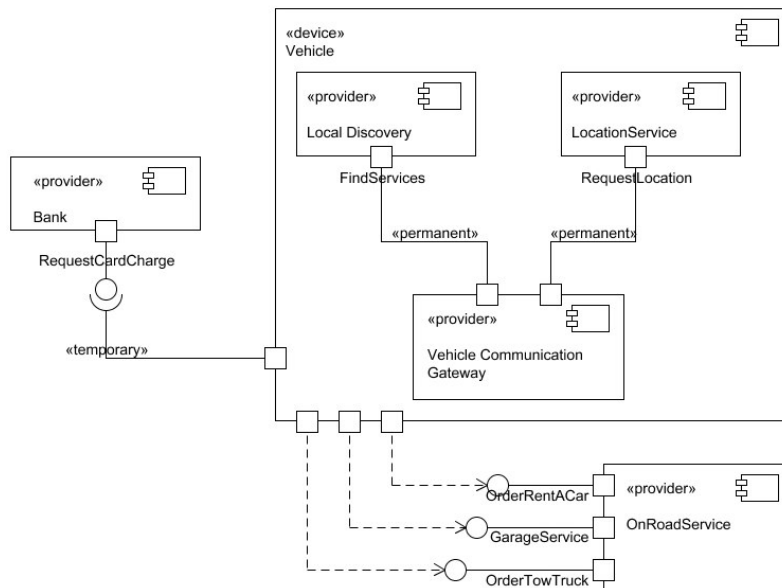


Figure 2: Car components modelled with UML extension for SOA before service publishing

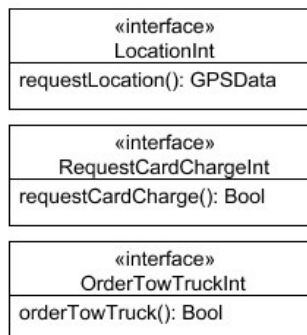


Figure 3: Interfaces of the services

Bocchi and Bustos [BB2007] present a SRML model for the business process of the Low Oil Scenario, which is part of the car repair scenario. There are two SRML modules created: The one assumes reliable communication between the components, such as GPS, On Road Repair Service, etc. The second module covers the cases where the communication is somehow broken or not available all the time. For example there are situations where a GPS receiver cannot provide any location data (e.g. when a car passes a tunnel). Then the common workflow cannot be accomplished and an alternative procedure must be chosen.

3.2 Modelling Behavioural Aspects

The most interesting aspect when modelling the behaviour of a service-oriented system is the workflow describing the orchestration of services. Modelling orchestration of services includes specifying non-functional properties of services such as performance and resource consumption, and modelling transactional business processes that may require a very long period in order to complete. As discussed above, the key technique to handle long running transactions is to install compensations, which are not directly available in UML.

3.2.1 Modelling Orchestration

In the modelled business process of the on road car repair scenario the orchestration is triggered by an engine failure or a sensor signal like low oil level. The process starts with a request from the Orchestrator to the Bank to charge the driver's credit card with the security deposit payment, which is modelled by an asynchronous UML action RequestCardCharge for charging the credit card the number of which is provided as output parameter of the UML stereotyped call action. Output pins have an arrow pointing away from the action; input pins have an arrow pointing toward the action. In parallel to the interaction with the bank the orchestrator initiates an asynchronous interaction to get the current position of the car from the GPS service. The current location is modelled as input to a receive interaction and subsequently used by the FindServices interaction which retrieves a list of services. For the selection of services the Orchestrator synchronises with the Reasoner to obtain the most appropriated services. Service ordering is modelled by the UML actions OrderGarage, OrderTowTruck and RentACar following a sequential and parallel process, respectively. Figure 5 shows an UML activity diagram of the orchestration of services in the on road car repair scenario. We use stereotyped UML actions indicating the type of interactions (callAsync, callSync, receive, reply) and model input and output parameters of the interactions with UML pins, which are not associated to any transition in the UML activity diagram. Interactions match operations of required and provided interfaces of the services. Services are defined as ports of UML components.

Bocchi and Bustos focus in the orchestration section of their report on the communication between the different components. Figure ??? shows a code fragment of the transition called 'Talert', which manages the different timeouts in the system. Timeouts occur while connecting to other components of the system, when having errors at sending messages or while waiting for the 'orderOnRoadService'. To know which type of interaction is currently associated with the timer, the local variable 'phase' is defined.

```

transition Talert
  triggeredBy now=timer
  effects phase=connect  $\supset$  bl'=bl  $\wedge$  sl'=sl+1  $\wedge$  s'=1
     $\wedge$  timer'=now+ctime
     $\wedge$  phase=sendError  $\supset$  bl'=bl+1  $\wedge$  sl'=1  $\wedge$  s'=1
       $\wedge$  timer'=now+ctime  $\wedge$  phase'=connect
     $\wedge$  phase=orderOnRoad  $\supset$  s'=5
       $\wedge$  callService(sendError[diagnostic])
  sends phase=orderOnRoad  $\supset$  connect[bl'][sl']!

```

Figure 4: SRML: A transition managing timeouts

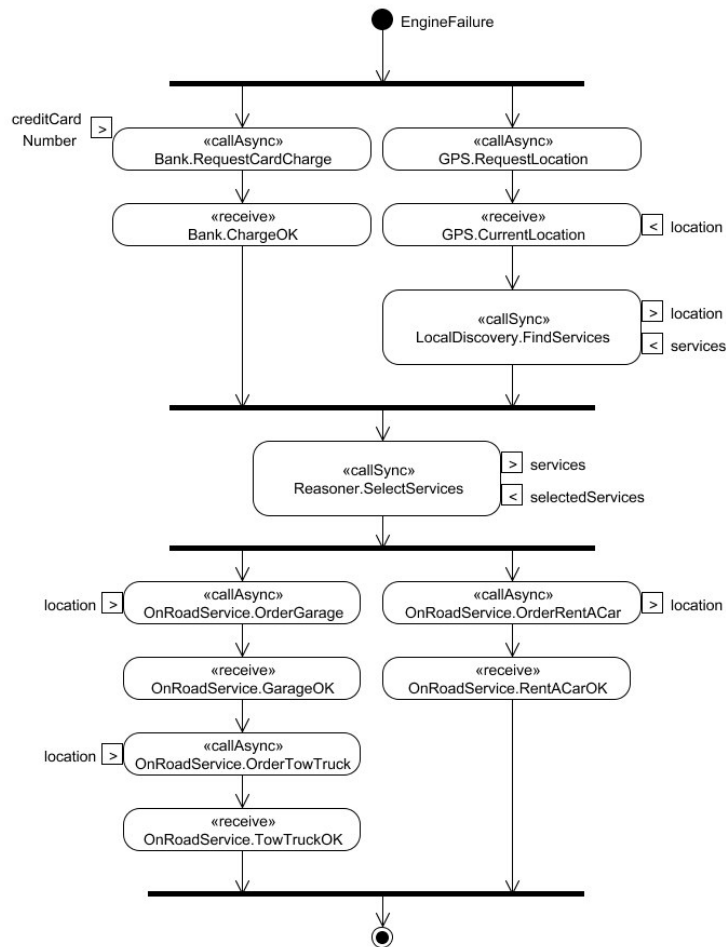


Figure 5: Orchestration in the on road car repair scenario

An approach to define a goal-based orchestration language is introduced by van Riemsdijk in [BW2007], which builds on research on goal-oriented agent programming, orchestration and semantic web services. The described approach of orchestration makes use of semantic matchmaking. The language is formally defined and analysed by using operational semantics. As scenario to show the approach the car repair scenario was chosen.

3.2.2 Modelling Compensation

The other main focus when modelling services lies on the specification of an appropriate transactional business process. Such a business process contains both forward actions and compensations. As UML activity diagrams lack such compensations, we defined in a first approach a set of modelling primitives and corresponding stereotypes for UML activity diagrams with a Saga-like semantics. Saga is an executable activity node that may have subordinate nodes as an ActivityGroup with the ability to compensate long running transactions. – CompensableAction specialises UML Action to own exactly one pair of actions (forward action and compensation action). To provide a more intuitive representation, both forward and compensation actions are drawn separated by a line within CompensableAction instances, although this is not completely UML compliant. The metamodel depicted in Fig. 7 shows how these compensation elements are related to UML elements StructuredActivityNode, ActivityNode and Action.

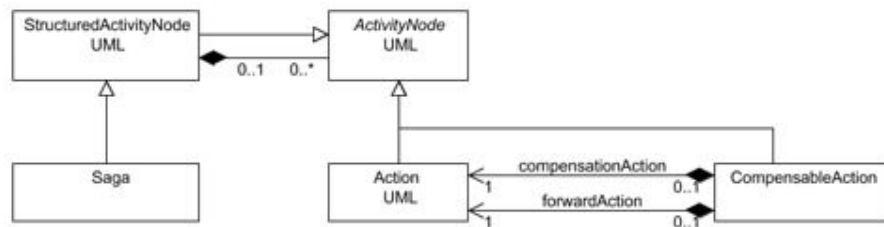


Figure 6: UML extension for sagas

With these extensions, the orchestration for the car repair scenario can be compactly formulated (Figure 7). In the modelled business process, the driver's credit card is charged with the security deposit payment, which will be revoked if ordering the services failed. Then, a garage appointment is searched for. The appointment with the garage will give coordinates to tow the broken down car to, and also a location constraint that restricts the car rental agency that may be ordered. If ordering the car rental fails, the overall process does not fail, as the activity is enclosed in a sub-transaction. However, if ordering a tow truck fails the garage appointment has to be cancelled as well. For this reason, the orchestrator will try to order a tow truck service until either no more service offers are found or the ordering succeeds. If ordering a tow truck fails the rental car delivery will be redirected to the driver's actual location.

It is obviously possible to model the same orchestration with a plain UML activity diagram, and handle compensations as exceptions (Figure 8). This requires explicit programming of the compensations and the conditions under which they are executed. In addition to actions, activities and control nodes, the specification requires an InterruptibleActivityRegion in order to terminate all active and pending activities of the region in case an interruption occurs. Even in this simple scenario, this approach requires the verification of three conditions. For larger scenarios the diagram's complexity will increase and its usefulness will decrease rapidly. Furthermore, it is difficult to explicitly model the silent failure of the car rental.

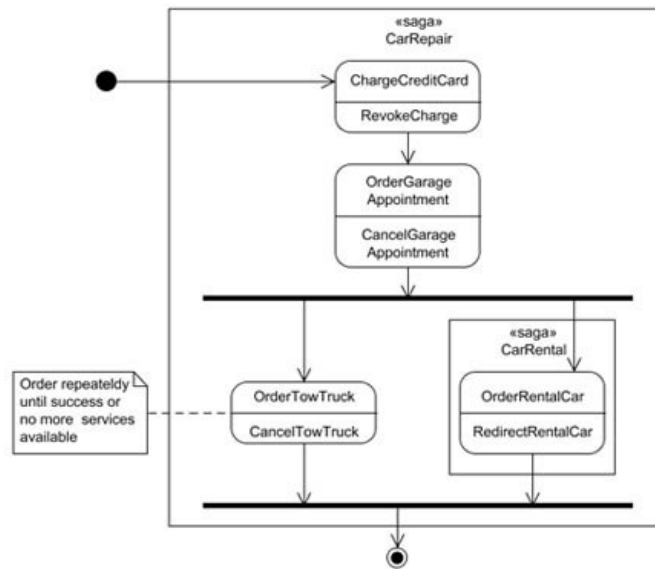


Figure 7: Modelling car repair workflow with UML extension for sagas

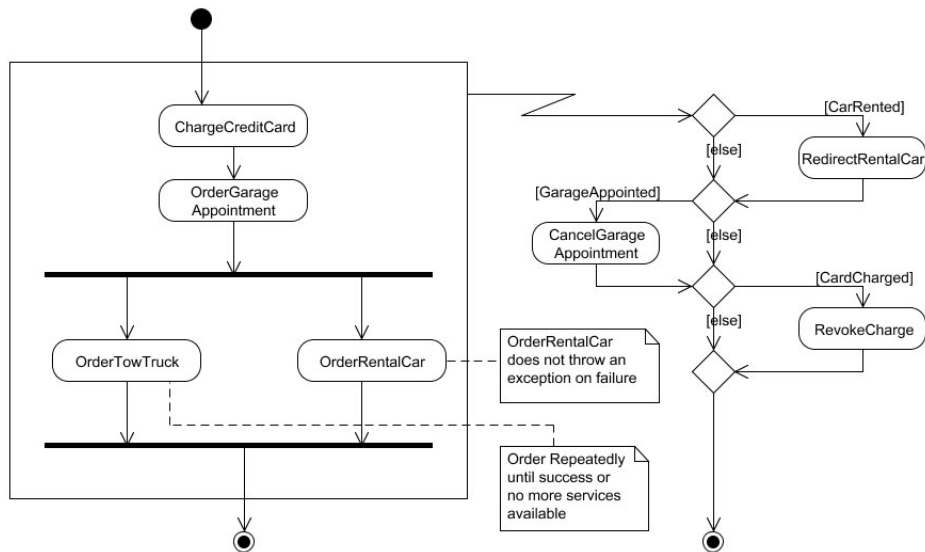


Figure 8: Modelling car repair workflow with plain UML activity diagram

We replaced this Saga-like approach by a more generic one that allows also to define complex compensation handlers.

For a detailed UML specification using the SENSORIA profile see the report ‘Automotive Case Study: UML Specification of On Road Assistance Scenario’ [Koch2007].

3.2.3 Modelling the Communication Flow

A simplified architecture of the communication flow of the automotive case study is modelled using plain UML techniques. Figure 9 provides an overview of the components of the automotive system considered for the flow. The following actors can be identified for this scenario:

- Service Planning: software component that reacts to specific triggers (for example anomaly in sensor’s data) and activates the right services (for example In-car diagnostic service).
- Communication System: permits communication between car’s internal and external systems.
- In-car Diagnostic: integrated service of the vehicle, analyses data coming from sensors and produces a description of the fault.
- External Diagnostic Service: service supply by the vehicle constructor. Sensor’s data are sent to the server where they are analysed and compared to a database of known error.
- Server returns a detailed analysis.
- On Road Repair Service: external service that activates service of automatic repair.

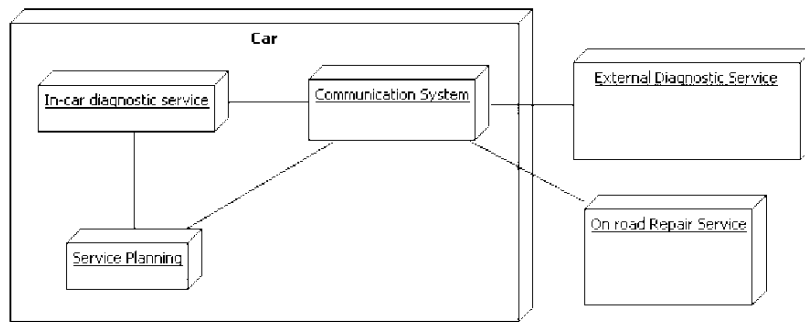


Figure 9: Deployment diagram of the car repair scenario

As shown in Figure 10, the Communication System can be regarded as the module, which controls the workflow of the whole scenario.

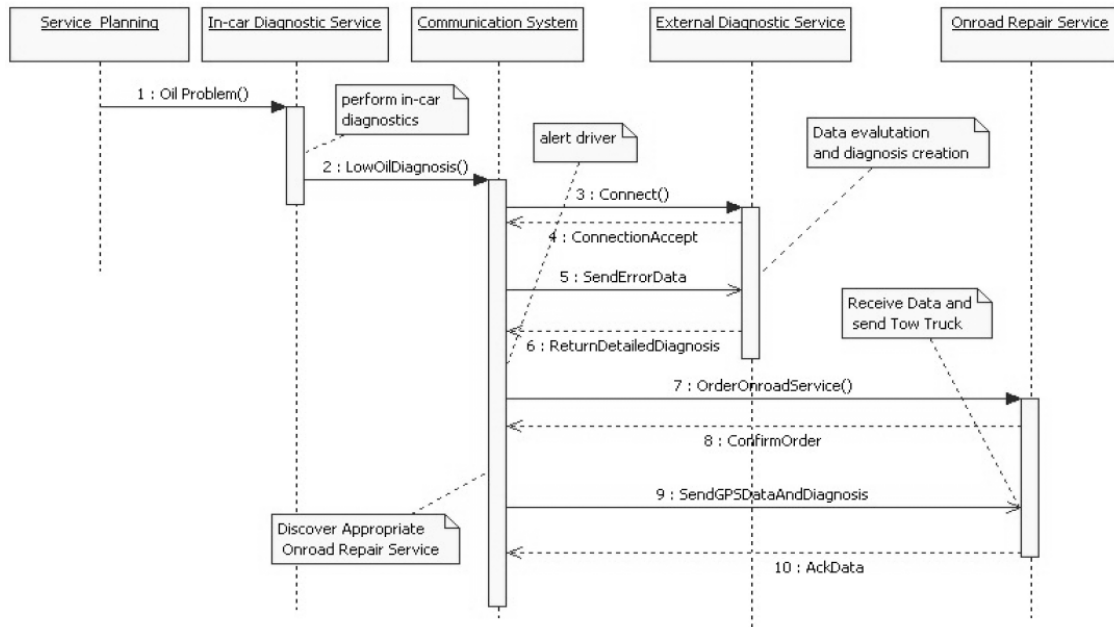


Figure 10: Sequence diagram of the car repair service

To describe the communication in depth Figure 11 shows the activity diagram of the module. First the communication system awaits the data from the In-car Diagnostic Server. The data contains information from the sensors which registered any abnormalities in the car system (e.g. low oil level). After collecting the data the communication system requests a connection to the External Diagnostic Service of the vehicle’s manufacturer. When the connection is established, all the collected data (containing sensor data, etc.) is handed over to the External Diagnostic Service. Now the communication system waits for the detailed diagnosis from

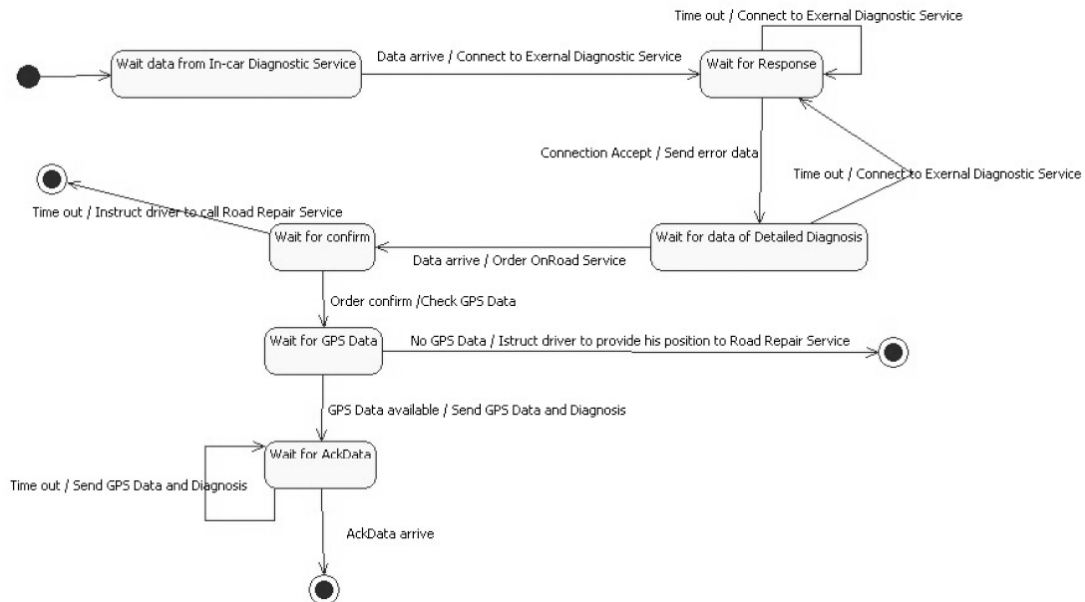
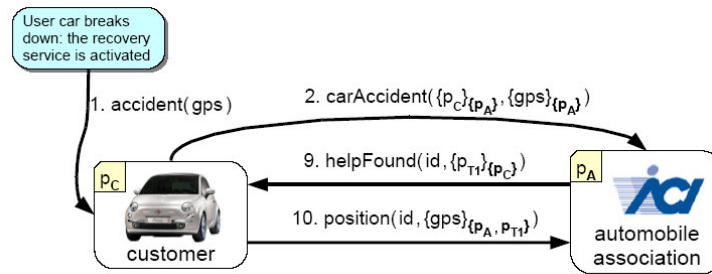


Figure 11: Activity diagram of the Communication System

the service. In company with the GPS data of the vehicle the diagnosis is now sent to a nearby towing station where the appropriate measures are taken and an acknowledgment is sent back to the communication system.

Tiezzi, Lapadula and Pugliese describes a type system using COWS for regulating data exchange between different entities [Ti2007]. The main goal was to get a 'type-based approach for expressing and forcing policies regulating the exchange of data among interacting services'. For demonstration purposes an enhanced Car Repair Scenario is created. This scenario is concerning security issues after suffering experiencing a car breakdown. The idea is that there is a group of trustworthy car drivers. After an accident occurred the automobile association server is contacted to get the closest trustworthy car. This car is asked to provide first aid to the possibly injured passengers. If the requested car denies, the second closest car is contacted and so on. Figure 12 show the step when the broken car gets the notification from the automobile association that a car which is willing to help is located (step 9). Subsequently, the broken car sends his position to the association, which forwards it to the trustworthy car.



$$\begin{aligned}
 & [\rho_{sys}] (\rho_{sys} \bullet O_{accident}! \langle gps \rangle \\
 & \quad | \{ \{ y_T \} \{ \rho_C \} \} \{ \{ y_{gps} \} \{ \rho_{sys}, p_A, y_T \} \} \rho_{sys} \bullet O_{accident} ? \langle y_{gps} \rangle . C) \\
 \\
 & C \triangleq (p_A \bullet O_{carAccident}! \langle \{ \rho_C \} \{ p_A \}, \{ y_{gps} \} \{ p_A \} \rangle \\
 & \quad | \{ \{ y_{id} \}^T \} (\rho_C \bullet O_{helpNotFound} ? \langle \rangle \\
 & \quad + \\
 & \quad \rho_C \bullet O_{helpFound} ? \langle y_{id}, y_T \rangle . p_A \bullet O_{position}! \langle y_{id}, \{ y_{gps} \} \{ p_A, y_T \} \rangle))
 \end{aligned}$$

Figure 12: Car security: Data exchange in COWS

In the context of the Sensoria project different core calculi have been defined. The different approaches developed for the work package 2, which aims to develop a methodology for describing service specifications and a discipline for their composition. The different calculi are: SOCK (a calculus for service oriented computing), COWS (a calculus for orchestration of web services), SCC (a service centred calculus), SC (signal calculus) and λ^{req} (a call-by-value calculus). To show the different features and workflows of the calculi, a model of the car repair scenario in each of the listed calculi is collected in a presentation [Pu2007]. Figure ??? shows an example for the λ^{req} calculus. The contract ϕ_L (loc) ensures that the requested tow truck is able to serve the location where the car break down happened. In parallel the contract ϕ_F (flt) ensures that the selected garage is able to fix the kind of faults *flt*, which caused the breakdown of the car. To guarantee the quality of the service, the surrounding contract ϕ_{BL} make sure that no black listed services are requested. To maintain the quality of the black list in turn, after accessing a service the user has the possibility to black list this service if it did not meet the demands of the user.

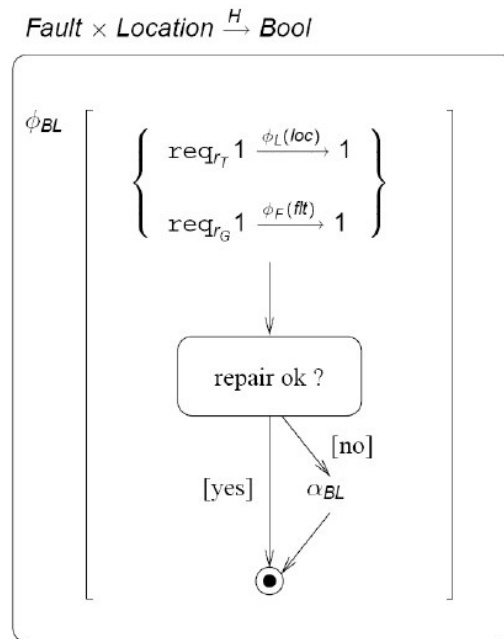


Figure 13: Car emergency service in λ^{req} calculus

Another core calculus, PSSC (Pipeline Service Centred Calculus) is proposed by Bruni [Br2007]. To show how to deal with the orchestration of services again the car repair scenario is used. As an example dependencies like ‘a garage must be found before looking for a tow truck’ are expressed in PSSC.

3.3 Quantitative and Qualitative Analysis

Qualitative and quantitative analysis methods for software systems aim at providing transparent support for the designer throughout the software construction phases based right on those notations used in development. However, proving the correctness of a design or measuring the performance, in general, relies on mathematical models and tool support that are not offered on the level of general software development notations, let alone using specialised extensions for particular domains. In SENSORIA, model transformations are employed to lift methods and tools from the well-founded, abstract, mathematical level to the concrete UML-based design level for service-oriented architectures. Furthermore, backward transformations project analysis results, delivered in terms of the underlying mathematical model, back to modelling notation.

To analyse the delivery of the services Nielson et al. [NN+2007] use an accident scenario, which is part of the Car Repair Scenario. For this scenario the correct delivery of messages in the implementation is proved.

3.3.1 Model Checking Orchestration

The orchestration of services, like in the extended UML activity diagram description in Figure 5, has to be implemented in an orchestrator. We transform the saga-based model into a conventional UML state machine model which details the handling of service allocation and compensation. Using model checking we can prove that the implementation model indeed preserves the compensation properties of the original UML model. The state machine in Figure 14 describes an implementation of the car repair workflow. It relies on a reasoner for choosing services. The general idea of the implementation is that every service needed is first requested from the reasoner (getService). If the reasoner delivers an offer (offer) in a certain amount of time (after(T)), the service first is reserved and ordered afterwards. If the reasoner misses the deadline, or the reservation or the ordering fail the orchestrator terminates.

In the latter cases the reasoner is informed of the failure (fail) such that it can avoid offering a failing service again. During the reservation phase the reasoner may send better offers for the requested service (offer). If a service can be reserved eventually, the reasoner is notified (reserved) and the offer is ordered. The compensation handling is done using a global compensation handler, like suggested by the UML activity diagram description in Figure 8.

For the services “charge credit card”, “order tow truck”, and “order car rental”, however, different implementation details are to be realised: On the one hand, the credit card charge need not be reserved; this is an implementation decision. Ordering the tow truck has been marked as to be done repeatedly until no more offers are available. Thus every possible offer has to be checked, that is, previous reservations may have to be compensated in the orchestration process (compensateReservation). Finally, the car rental may or may not fail without calling for overall compensation, as it is marked as a nested saga.

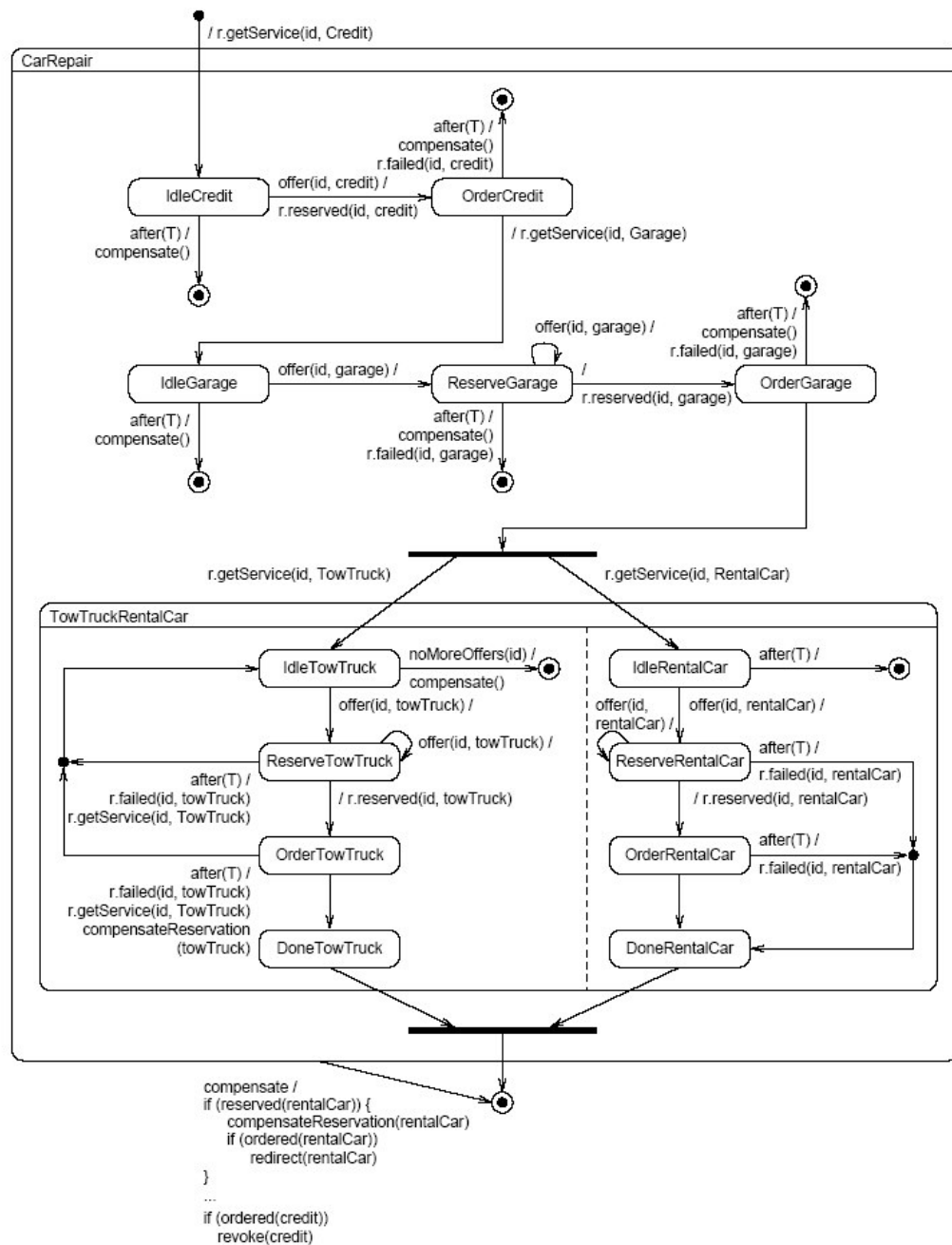


Figure 14:Implementation state machine for car repair workflow

In SENSORIA different tools are available for verifying the correct behaviour of the orchestrator implementation. The UML model checker UMC [?] (developed by ISTI) offers the verification of temporal properties described in the μ -calculus directly on UML state machine. The UML model checking tool Hugo/RT [?] (developed by LMU) translates UML state machines and collaborations into different off-the-shelf model checkers like Spin [?] and UPPAAL [?] and also supports Java and SystemC code generation. Here, we used the UPPAAL option of Hugo/RT. By instrumenting the model, which we will not detail here, we checked, e.g., that whenever the global final state is reached all services indeed have been ordered; if an order fails all orders and reservations which have been done up to the point of failure of getting a service—i.e., when one of the final states inside CarRepair is entered—are compensated. After behaviour verification, we have used Hugo/RT for generating Java code from the very same model that has been verified by UPPAAL.

4 Specification of the Airbag Scenario

In the accident assistance scenario, due to a head on collision, the vehicles airbag is deployed which triggers an automated message to the accident assistance server that contains the vehicle's GPS data, the vehicle identification number and a collection of sensor data like for example the number of seatbelts in use and impact sensors for critical vehicle parts. The safety system report the car's location to an accident report centre. This centre attempts to determine the severity of the accident and take appropriate actions.

The accident assistance server places a call to the driver's mobile phone. Due to his injuries, the driver is unable to answer the call. Based on the sensor data available to the accident assistance server, the severity of the accident is assessed and emergency services (police, ambulance) are alerted and provided with the vehicles location. Approaching vehicles are warned about the accident ahead through wireless messages from the vehicle involved in the accident. Accident information is successively passed on to the next approaching vehicles.

A precondition is that the driver has subscribed to the accident assistance service available for all vehicles of the car manufacturer.

4.1 Modelling the scenario

Modelling orchestration of services as already said also includes specifying non-functional properties of services such as performance and resource consumption. We start with the modelling of the accident assistance scenario. The accident assistance scenario is concerned with road traffic accidents and dispatch of medical assistance to crash victims. Drivers wishing to use the service must have in-car GPS location tracking devices with communication capabilities and have pre-registered their mobile phone information with the service. If a road traffic accident occurs, the deployment of the car airbag causes the on-board safety system to report the car's current location (obtained by GPS) to a pre-established accident report endpoint which in turn attempts to call the registered driver's mobile phone. If there is no answer to the call then medical assistance is dispatched to the reported location of the car (presuming that the driver has been incapacitated by injuries sustained in the accident).

We model this scenario in UML as state machine; to represent quantitative aspects (e.g., answer time) we use stereotypes to attach rates to transitions, see **Fehler! Verweisquelle konnte nicht gefunden werden.** Figure 15 explains the meaning of the rates.

Rate	Value		Meaning
	min	max	
r_1	600.0	600.0	an airbag deploys in 1/10 of a second
r_2	2.0	10.0	the car can transmit location data in 6 to 30 seconds
r_3	0.5	1.5	it takes about one minute to register the incoming data
r_4	1.5	2.5	it takes about thirty seconds to call the driver's phone
r_5	1.0	60.0	give the driver from a second to one minute to answer
r_6	0.25	3.0	vary about one minute to decide to dispatch medical help
r_7	1.0	1.0	arbitrary value — the driver is now awaiting rescue

Figure 15: Table of minimum and maximum values of the rates **Fehler! Verweisquelle konnte nicht gefunden werden.** All times are expressed in minutes. Thus a rate of 1.0 means that something happens once a minute (on average). A rate of 6.0 means that the associated activity happens six times a minute on average, or that its mean or expected duration is ten seconds, which is an equivalent statement.

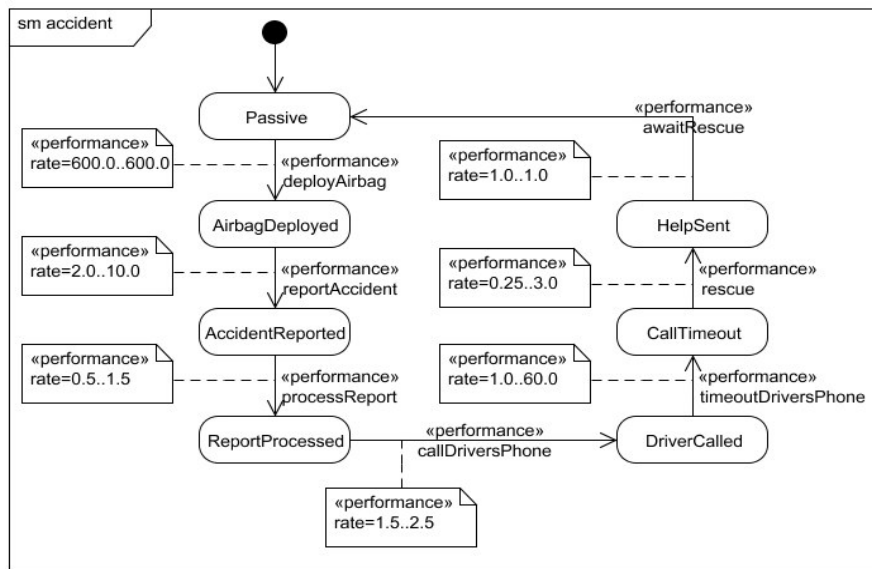


Figure 16: State machine of the accident assistance scenario

4.2 System overview

The system consists of two different kinds of modules: There are modules integrated in the vehicle and the modules outside the vehicle: An overview of the modules is presented in the deployment diagram in Fehler! Verweisquelle konnte nicht gefunden werden..

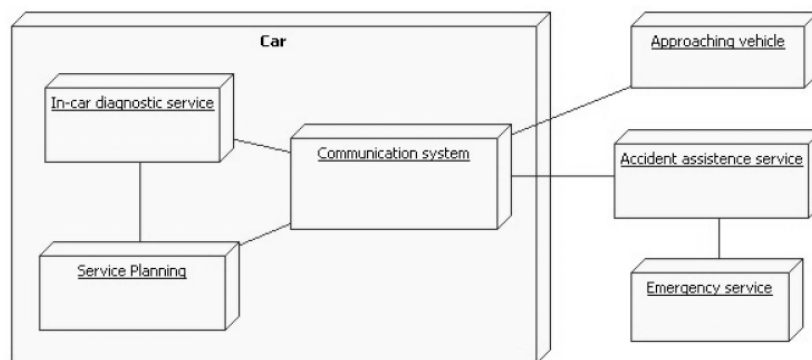


Figure 17: System overview of the airbag scenario

Afterwards the different actors in the system are described in detail.

- Service Planning: software component that reacts at specific triggers (for example anomaly in sensor’s data) and activates the right services (for example In-car Diagnostic Service).

- Communication System: permits communication between car's internal and external systems.
- In-car Diagnostic: integrated service of the vehicle, analyses data coming from sensors and produces a description of the fault.
- Accident Assistance Service: external service that receives GPS and sensor data from the Communication System and contacts the Emergency Service. Alerts approaching vehicles.
- Emergency Service: receives the call of the Accident Assistance Service and alerts emergency services (police, ambulance).
- Approaching Vehicles: vehicles approaching the accident location, they are warned about the accident ahead through wireless messages.

4.3 Workflow in the airbag scenario

The following diagrams illustrate the workflow between the different components in the airbag scenario. First of all, in **Fehler! Verweisquelle konnte nicht gefunden werden.** the "normal" run of an airbag deployment is shown. On the left hand side the in-car modules Service Planning, InCar Diagnostic Service and the Communication System are placed. The Communication System acts as an interface to the in-car systems and the modules outside the range of the vehicle and can be regarded as the central component in the airbag scenario.

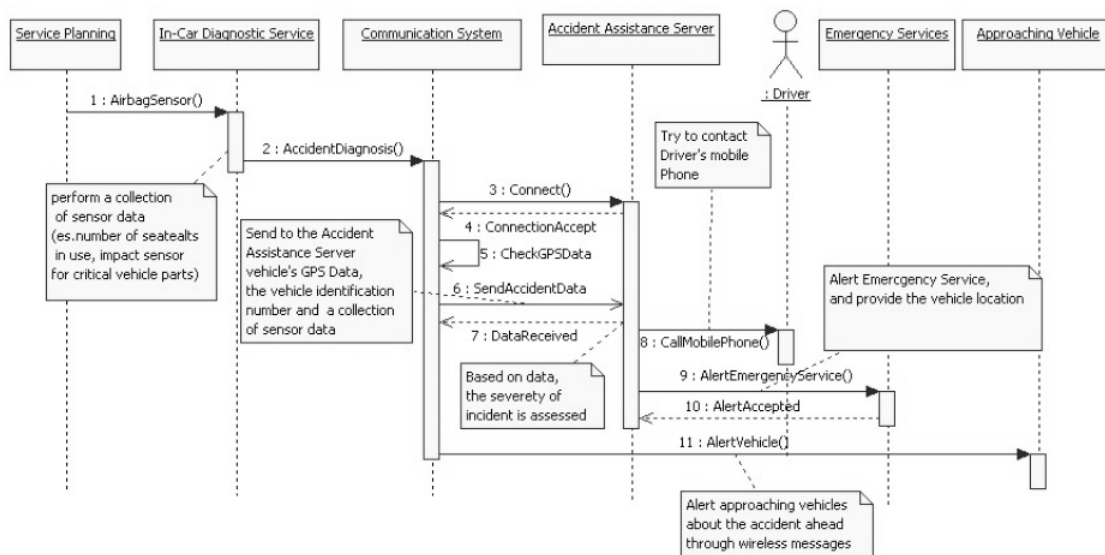


Figure 18: Sequence diagram for the airbag scenario

4.4 Activities of the communication system

To describe the communication in depth **Fehler! Verweisquelle konnte nicht gefunden werden.** shows the activity diagram of the module. First the communication system awaits the data from the InCar Diagnostic Server. The data contains information collected different sensors in the vehicle. After collecting the data the communication system requests a connection to the Accident Assistant Server of the vehicle's manufacturer. When the connection is established, all the collected data (containing sensor data, GPS data, etc.) is handed over to the Accident Assistant Server. Now the communication system waits for the acknowledgment of the server and then starts the procedure to alert vehicles, which are approaching the damaged car.

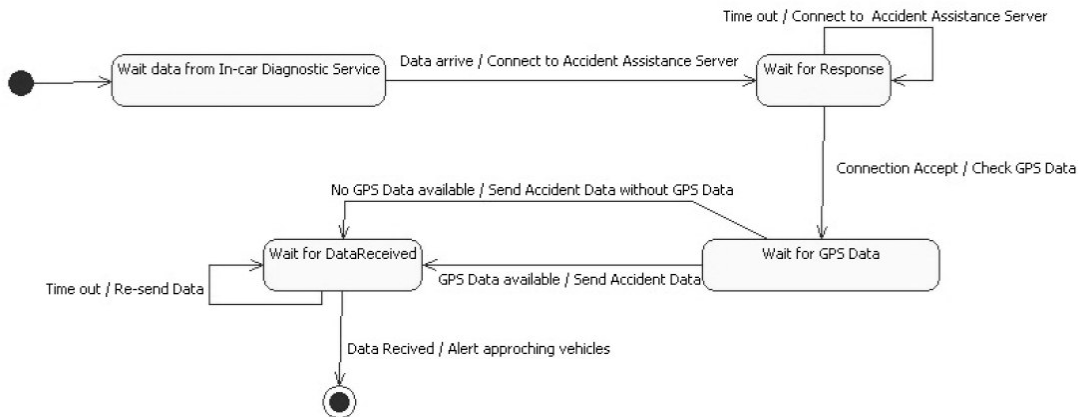


Figure 19: Activity diagram for the communication system in the airbag scenario

Clark and Gilmore use the airbag scenario for a work, which focuses on the quality of a provided service [CG2006]. For the quantitative analysis of the service the time from deploying the airbag and the time when the ambulance is sent to the possibly injured passengers of the broken car is taken. Therefore the service is

$$Car_1 \stackrel{def}{=} (airbag, r_1).Car_2$$

$$Car_2 \stackrel{def}{=} (reportToService, r_2).Car_3$$

$$Car_3 \stackrel{def}{=} (processReport, r_3).Car_4$$

Figure 20: Reporting the accident

modelled in the PEPA algebra. Figure ?? shows the first phase of the scenario, were the accident is registered at the remote service center. After deploying the airbag, a report with the car data is sent to this center and the report is processed. The next phase is an attempted communication with the driver. This is to determine if medical help has to be sent to the location of the breakdown. If the river of the car does not answer any calls from the service center, this leads to a rescue routine, where medical staff is sent to the injured driver. (Figure ???).

$$Car_4 \stackrel{def}{=} (callDriversPhone, r_4).Car_5$$

$$Car_5 \stackrel{def}{=} (timeoutDriversPhone, r_5).Car_6$$

$$Car_6 \stackrel{def}{=} (rescue, r_6).Car_7$$

$$Car_7 \stackrel{def}{=} (awaitRescue, r_7).Car_1$$

Figure 21: Triggering the rescue routine

This model is then compiled with the PEPA compiler ipc into a stochastic Petri net, which is analysed with a Markov chain analyser.

Another work uses the airbag scenario to present the Mobile Continuous Stochastic Logic(MoSL) [NK2007+], which is described as follows: MosL is

- a temporal logic that permits describing the dynamic evolution of the system,
- both action- and state-based,
- a real-time logic that permits the use of real-time bounds in the logical characterisation of the behaviours of interest,
- a probabilistic logic that permits expressing not only functional properties, but also properties related to performance and dependability aspects, and, finally,
- a spatial logic that addresses the spatial structure of the network for the specification.

5 Specification of the Reconfigurable Route Planning Scenario

Steven and John are on their way to Italy in separate cars, both have their wife and children on board since the two families want to spend their holidays together. John has entered the destination into his navigation system which is in “planning”-mode and thus is calculating and providing the best route during the travel. To make sure both cars take the same route, Steven’s navigation system is configured to be in “convoy”-mode, which means that the vehicle’s route is guided by another vehicle (in this case John’s). So Steven’s navigation system just receives route planning information from John’s instead of performing route planning itself.

Due to a collision on the highway some kilometres in front both the “planning”-mode and the “convoy”-mode get disabled because they get overridden by the “detour”-mode. In this mode the Highway Emergency System takes control of the user interface of the navigation system and sends a warning and an alternative route to avoid the scene of accident.

The first two cases, i.e. the planning of a route by the driver on his own or following another car in a convoy is presented in the following specification. How can the system distinguish between these two possibilities?

There is a proposal to use different “modes” to manage the tasks from Howard Foster: http://www.pst.ifi.lmu.de:8080/Sensoria/DOWNLOAD/meetings/0703_london/20070322-6.1b-selfmanagement.pdf a mode is defined as follows:

“A mode abstracts a specific set of services that must interact for the completion of a specific subsystem task”

There are two different modes in the scenario: Planning mode and the convoy mode. In each mode the system accesses different sets of services as you can see in Figure 22 & Figure 23. In the planning mode the vehicle relies on public services and plans the route on its own.. In the convoy mode the vehicle is connected to the planning module of the vehicle it wants to be guided of.

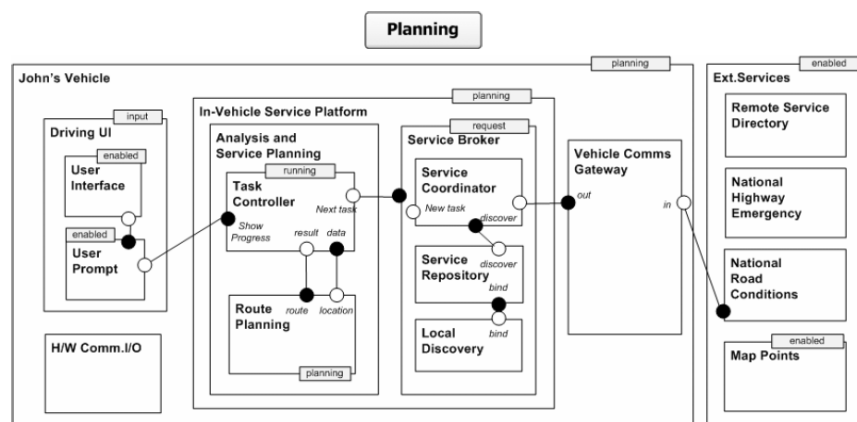


Figure 22: Vehicle in the planning mode

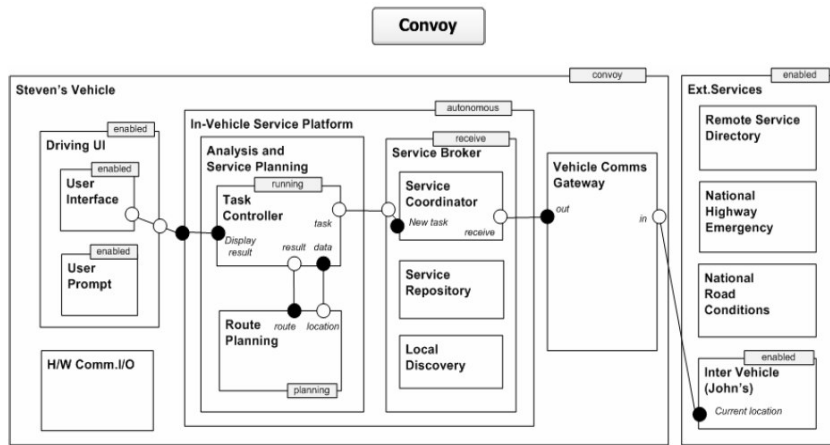


Figure 23: Vehicle in the convoy mode

How to get a route while being in the planning mode is shown in the diagram Figure 23. The user provides his preferred destination to the TaskController, which interacts with the RoutePlanning module. After querying the proper services the selected route is given back to the user.

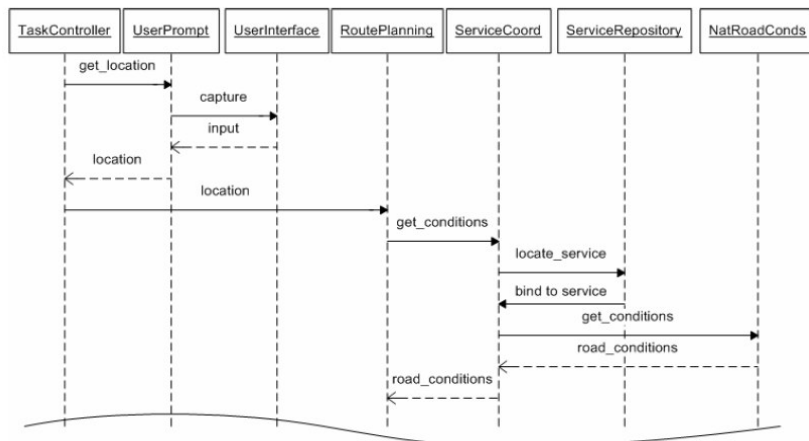


Figure 24: Retrieval of a route in the planning mode

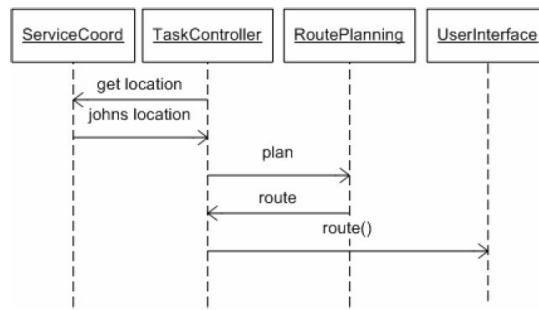


Figure 25: Retrieval of a route in the convoy mode

According to Figure 25 the retrieval of the route changes when the vehicle is in convoy mode. To get the route now the destination of the buddy is used (“johns location” in the diagram). With this destination the RoutePlanning module calculates route data and provides it to the user interface.

6 Specification of the Road Sights Scenario

The driver has subscribed to the dynamic sight service offered by the car company. The vehicles GPS coordinates are automatically sent to the dynamic sight server at regular intervals, so the vehicles location is known within a specified radius. Based on the driver’s preferences that were given at the beginning of the trip, the dynamic sight server searches a sight seeing database for appropriate sights and displays them on the in-car map of the vehicles’ navigation system. The driver clicks on sights he would like to visit which results in the display of more detailed information about this specific sight (e.g. opening times, guidance to parking etc.).

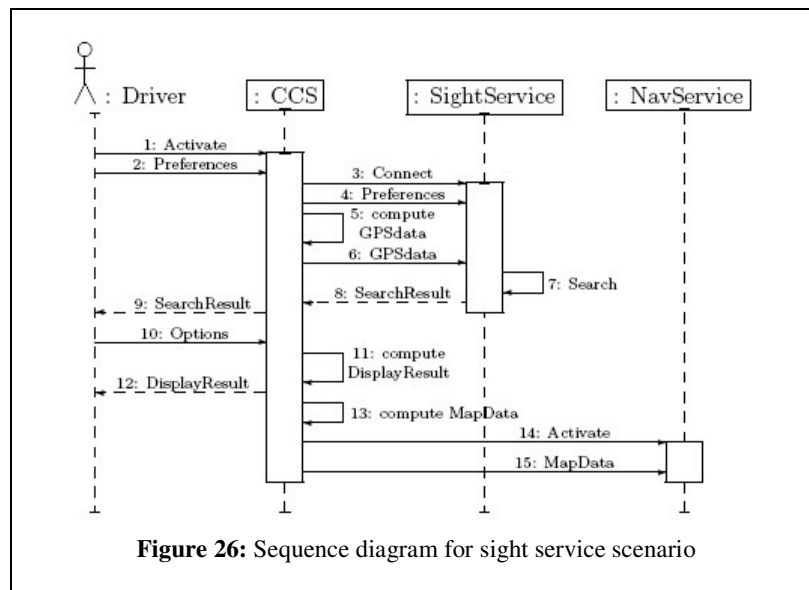


Figure 26: Sequence diagram for sight service scenario

A scenario similar to the Road Sights Scenario is described by Nielson [NN2007]: A car is equipped with an info system consisting of a GPS module and a message board. The car permanently sends the position of the car via an wireless link. Over the message board he can retrieve information like news. Concerning the security domain there are two questions: How to prevent getting unsolicited information (e.g. ads, spam, etc.) and how

to avoid that the position data sent over the air is not received by non authorized entities? The idea is now to develop a static program analysis for the pi-calculus and to show how it can be used to ensure privacy guarantees. As part of the paper the Car Communication System (CCS) is implemented as the following process in SSCC. Thereby the notation $\langle \text{action} \rangle$ represents an internal action of the system. The numbers on the right correspond to the numbers of the sequence diagram in **Fehler! Verweisquelle konnte nicht gefunden werden.**

```

CCS  $\Rightarrow$  (Preferences). // 1:, 2:
  (SightService  $\Leftarrow$  Preferences. // 3:, 4:
     $\langle$ compute GPSdata $\rangle$ . // 5:
    GPSdata. // 6:

    (SearchResult). // 8:
    feed SearchResult)

 $\rangle^1$  SearchResult  $\rangle$ 
SearchResult. // 9:
(Options). // 10:
 $\langle$ compute DisplayResult $\rangle$ . // 11:
DisplayResult. // 12:
 $\langle$ compute MapData $\rangle$ . // 13:
(NavSystem  $\Leftarrow$  MapData) // 14:, 15:

```

Figure 27: Implementation in SSCC

Another use of the road sights scenario is described in [CM+2007], where the way is shown, how the scenario can be represented with SSCC (Stream-based Service Centred Calculus).

7 Specification of the Dinner Break Scenario

Paul is very hungry since he is driving without any food for 5 hours. So he activates the dinner service and enters a pizzeria as desired restaurant type and a price range between five and ten euros per meal. The navigation system displays a collection of nearby restaurants which match the preferred settings. Paul chooses the option to check for available seats in the participating local restaurants and as a result the map displays only restaurants with available tables. Paul chooses “Tony’s Pizza” and gets his reservation acknowledged. The way to the restaurants parking slot is now displayed on the navigation system map.

As described before for the road sights scenario, [CM+2007] provides a way to represent the dinner break scenario with SSCC as well.

8 Bibliography

- [BH+2006] Luciano Baresi, Reiko Heckel, Sebastian Thöne, and Daniel Varro: Style-based Modelling and Refinement of Service-oriented Architectures. *Softw. Sys. Model.*, 2006. To appear.
- [Br2007] Roberto Bruni: PipelineSCC at Work, Sensoria General Meeting, Leicester 2007
- [BB2007] Laura Bocchi, Jose Bustos: Automotive Case Study: LowOil Scenario; A SRML Model, Sensoria General Meeting 2007
- [Pu2007] Rosario Pugliese: An Introduction to the Core Calculi of WP2, Sensoria THEME 2and Case Studies Workshop, Paderborn, March 2007
- [Ti2007] Francesco Tiezzi: Regulating data exchange in COWS specifications, Sensoria General Meeting, Leicester 2007
- [WG+2006] Martin Wirsing, Stephen Gilmore, Allan Clark, Matthias Hölzl, Alexander Knapp, Nora Koch, Andreas Schroeder: Semantic-Based Development of Service-Oriented Systems, in *FORTE 2006, LNCS 4229, pp.24-25, 2006*
- [BW2007] Birna van Riemsdijk, Martin Wirsing: Using Goals for Flexible Service Orchestration: A First Step,

- [CG2006] SOCASE 2007,
Allan Clark, Stephen Gilmore: Evaluating Quality of Service for Service Level Agreements, in *Proceedings of FMICS and PDMC 2006*, Springer Verlag, LNCS 4346, pages 181-194, August 2006
- [NK2007+] R. de Nicola, J.P.Katoen, D. Latella, M. Loreti, M. Massink: Deliverable 4.2.a: Stochastic Logics, Sensoria Project
- [NN2007] Hanne Ries Nielson, Flemming Nielson: A flow-sensitive analysis of privacy properties , to be published in CSF 2007
- [CM+2007] Luis Cruz-Filipe, Francisco Martins, Vasco Vasconcelos: The Automotive Case Study in the Sensoria Core Calculi, June 2007
- [NN+2007] Flemming Nielson, Hanne Riis Nielson, Jörg Bauer, Christoffer Rosenkilde Nielsen, Henrik Pilegaard: Relational Analysis for Delivery of Services, Sensoria General Meeting Leicester, June 2007
- [Koch2007] Nora Koch: Automotive Case Study: Specification of On Road Assistance Scenario, August 2007
- [WB+2007] Martin Wirsing, Laura Bocchi, Allan Clark, Jose Luiz Fiadeiro, Stephen Gilmore, Matthias Hölzl, Nora Koch, Rosario Pugliese: SENSORIA: Engineering for Overlay Computers, MIT book, June 2007