A COWS Specification of an Automotive Case Study

Companion Technical Report to "*A Calculus for Orchestration of Web Services*"
submitted to *Journal of Applied Logic*

Authors: Rosario Pugliese and Francesco Tiezzi

Date: March 16, 2011
Institute: Dipartimento di Sistemi e Informatica, Università degli Studi di Firenze

1 An automotive case study

We introduce a significant case study [2] in the area of automotive systems defined within the EU project SENSORIA [5]. We consider a scenario where vehicles are equipped with a multitude of sensors and actuators that provide the driver with services that assist in conducting the vehicle more safely. Driver assistance systems become automatically operative when the vehicle context renders it necessary. Due to the advances in mobile technology, automotive software installed in the vehicles can contact relevant specific services to deal with driver's necessities.

Specifically, let us consider the case in which, while a driver is on the road with her/his car, the vehicle's *sensors monitor* reports a severe failure, which results in the car being no longer driveable. The car's *discovery* system then identifies garages, car rentals and towing truck services in the car's vicinity. At this point, the car's *reasoner* system chooses a set of adequate services taking into account personalised policies and preferences of the driver, e.g. balancing cost and delay, and tries to order them. To be authorised to order services, the car's system has to deposit on behalf of the car owner a security payment, which will be given back if ordering the services fails. Other components of the in-vehicle service platform involved in this assistance activity are a *GPS* system, providing the car's current location, and an *orchestrator*, coordinating all the described services.

An UML-like activity diagram of the orchestration of services using UML4SOA, an UML Profile for service-oriented systems [3], is shown in Figure 1. The orchestrator is triggered by a signal from the sensors monitor (concerning, e.g., an engine failure) and consequently contacts the other components to locate and compose the various services to reach its goal. The process starts with a request from the orchestrator to the *bank* to charge the car owner's credit card with the security deposit payment. This is modelled by the UML action CardCharge for charging the credit card whose number is provided as an output parameter of the action call. In parallel to the interaction with the bank, the orchestrator requests the current location of the car from the car's internal GPS system. The current location is modelled as an input to the RequestLocation action and subsequently used by the FindServices interaction which retrieves a list of services. If no service can be found, an action to compensate the credit card charge will be launched. For the selection of services, the orchestrator synchronises with the reasoner service to obtain the most appropriate services.

Service ordering is modelled by the UML actions OrderGarage, OrderTowTruck and RentCar. When the orchestrator makes an appointment with the garage, the diagnostic data are automatically transferred to the garage, which could then be able, e.g., to identify the spare parts needed to perform the repair. Then, the orchestrator makes an appointment with the towing service, providing the GPS data of the stranded vehicle and of the garage, to tow the vehicle to the garage. Concurrently, the orchestrator makes an appointment with the rental service, by indicating the location (i.e. the GPS coordinates either of the stranded vehicle or of the garage) where the car will be handed over to the driver.

The workflow described in Figure 1 models the overall behaviour of the system. Besides interactions among services, it also includes activities using concepts developed for long running business transactions (in e.g. [1, 4]). These activities entail fault and compensation handling, kind of specific activities attempting to reverse the effects of previously committed activities, that are an important aspect of SOC applications. According to UML4SOA Profile, the installation of a compensation handler is modelled by an edge stereotyped ≪compensationEdge≫, and its activation by an activity stereotyped ≪compensate≫. Since each compensation handler is associated to a single UML activity, we omit drawing the enclosing 'scope' construct. Moreover, we use dashed boxes to represent compensation handlers. Specifically, in the considered scenario:

- the security deposit payment charged to the car owner's credit card must be revoked if either the discovery phase does not succeed or ordering the services fails, i.e. both garage/tow truck and car rental services reject the requests;

- if ordering a tow truck fails, the garage appointment has to be cancelled;

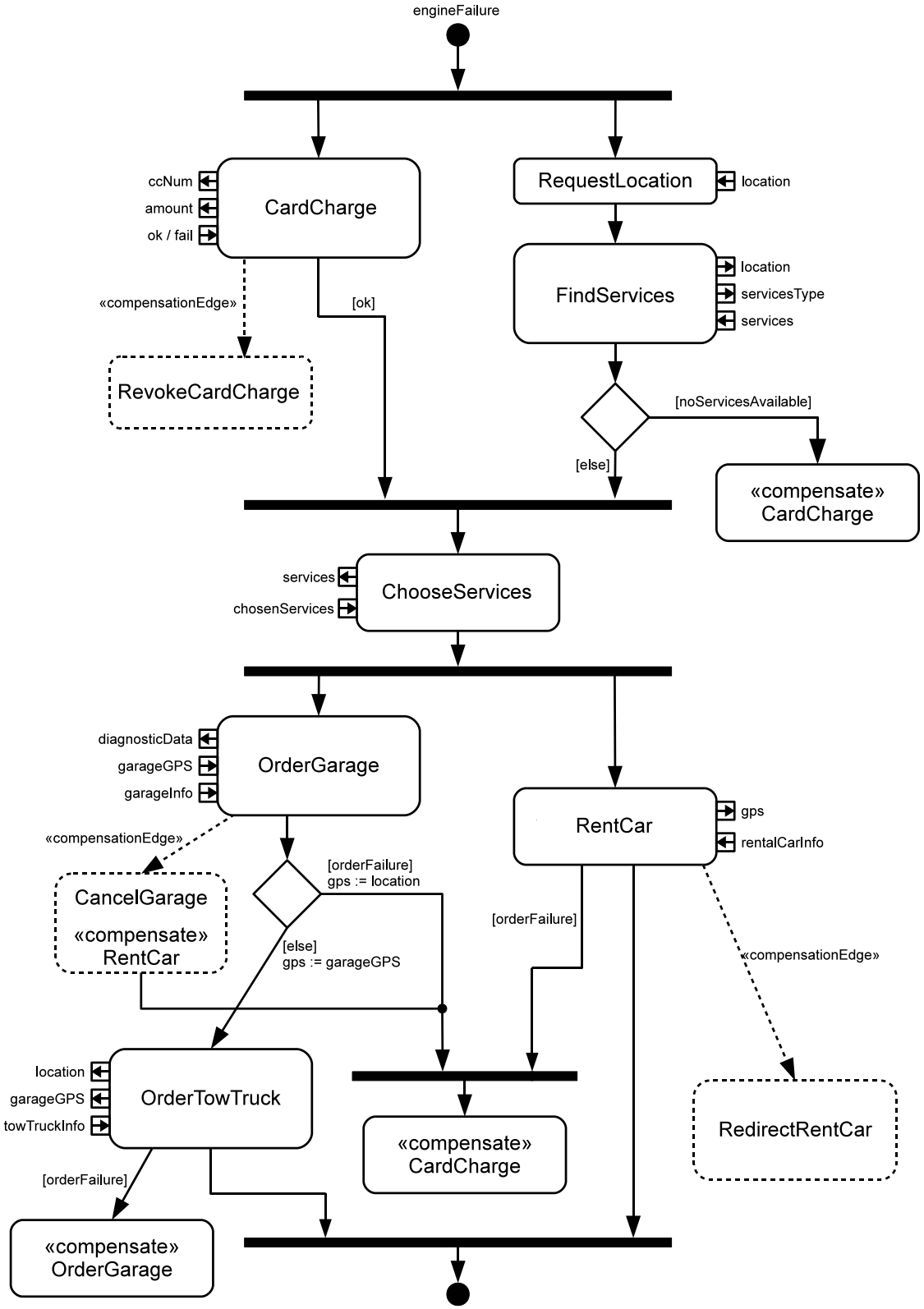

Figure 1: Orchestration in the automotive scenario

- if ordering a garage fails or a garage order cancellation is requested, the rental car delivery has to be redirected to the stranded car's actual location;
- instead, if ordering the car rental fails, it should not affect the tow truck and garage orders.

These requirements motivate the fact that ordering garage/tow truck and renting a car are modelled as activities running in parallel.

2 Complete specification of the automotive case study

The COWS term representing the overall automotive scenario is:

$$[p_{car}]\,(\,\mathit{SensorsMonitor} \mid \mathit{GpsSystem} \mid \mathit{Discovery} \mid \mathit{Reasoner} \mid \mathit{Orchestrator}\,) \mid \mathit{Bank} \mid \mathit{OnRoadRepairServices}$$

All services of the in-vehicle platform share a private partner name p_{car}, that is used for intra-vehicle communication and is passed to external services (e.g. the bank service) for receiving data from them.

When an engine failure occurs, a signal (raised by *SensorsMonitor*) triggers the execution of the *Orchestrator* and activates the corresponding 'recovery' service. *Orchestrator*, the most important component of the in-vehicle platform, is

$$[x_{carData}, x_{ts}]\,(p_{car} \bullet o_{engineFailure}?\langle x_{ts}, x_{carData}\rangle.s_{engfail} \;+\; p_{car} \bullet o_{lowOilFailure}?\langle x_{ts}, x_{carData}\rangle.s_{lowoil} \;+\; \ldots)$$

This term uses the choice operator $\_ + \_$ to pick one of those alternative recovery behaviours whose execution can start immediately. Notice that, while executing a recovery behaviour, *Orchestrator* does not accept other recovery requests. We are also assuming that it is reinstalled at the end of the recovery task.

The recovery behaviour $s_{engfail}$ executed when an engine failure occurs is

$$[p_{end}, o_{end}, x_{info}, x_{loc}, x_{list}, o_{undo}]\;(\,[k]\,(\,\mathit{CardCharge} \mid \mathit{FindServices}\,) \;\mid\; p_{end} \bullet o_{end}?\langle\rangle.\, p_{end} \bullet o_{end}?\langle\rangle.\, \mathit{ChooseAndOrder}\,)$$

$p_{end} \bullet o_{end}$ is a scoped endpoint along which successful termination signals (i.e. communications that carry no data) are exchanged to orchestrate execution of the different components. *CardCharge* corresponds to the homonymous UML action of Figure 1, while *FindServices* corresponds to the sequential composition of the UML actions RequestLocation and FindServices. The two terms are defined as follows:

$$\begin{aligned}
\mathit{CardCharge} \triangleq\;& p_{bank} \bullet o_{charge}!\langle p_{car}, ccNum, amount, x_{ts}\rangle \\
&\mid \{\!|\, p_{car} \bullet o_{resp}?\langle fail, x_{ts}, x_{info}\rangle.\, \mathbf{kill}(k) \\
&\quad + p_{car} \bullet o_{resp}?\langle ok, x_{ts}, x_{info}\rangle. \\
&\quad\quad (\, p_{end} \bullet o_{end}!\langle\rangle \\
&\quad\quad\quad \mid p_{car} \bullet o_{undo}?\langle cc\rangle.\, p_{car} \bullet o_{undo}?\langle cc\rangle.\, p_{bank} \bullet o_{revoke}!\langle x_{ts}, ccNum\rangle\,)\,|\!\}
\end{aligned}$$

$$\begin{aligned}
\mathit{FindServices} \triangleq\;& p_{car} \bullet o_{reqLoc}!\langle\rangle \\
&\mid p_{car} \bullet o_{respLoc}?\langle x_{loc}\rangle. \\
&\quad (\, p_{car} \bullet o_{findServ}!\langle x_{loc}, servicesType\rangle \\
&\quad\quad \mid p_{car} \bullet o_{found}?\langle x_{list}\rangle.\, p_{end} \bullet o_{end}!\langle\rangle \\
&\quad\quad + p_{car} \bullet o_{notFound}?\langle\rangle. \\
&\quad\quad\quad (\, \{\!|\, p_{car} \bullet o_{undo}!\langle cc\rangle \mid p_{car} \bullet o_{undo}!\langle cc\rangle\,|\!\} \mid \mathbf{kill}(k)\,)\,)
\end{aligned}$$

Therefore, the recovery service concurrently contacts service *Bank*, to charge the driver's credit card with a security amount, and services *GpsSystem* and *Discovery*, to get the car's location (stored in x_{loc}) and a list of on road services (stored in x_{list}). When both activities terminate (the fresh endpoint $p_{end} \bullet o_{end}$ is used to appropriately synchronise their successful terminations), the recovery service forwards the obtained list to service *Reasoner*, that will choose the most convenient services (see definition of *ChooseAndOrder*). Whenever services finding fails, *FindServices* terminates the whole recovery behaviour (by means of the kill activity $\mathbf{kill}(k)$) and sends two signals *cc* (abbreviation of 'card charge')

along the endpoint $p_{car} \bullet o_{undo}$. Similarly, if charging the credit card fails, then *CardCharge* terminates the whole recovery behaviour. Otherwise, it installs a compensation handler that takes care of revoking the credit card charge. Activation of this compensation activity requires two signals *cc* along $p_{car} \bullet o_{undo}$ and, thus, takes place either whenever *FindService* fails or, as we will see soon, whenever both garage and car rental orders fail.

ChooseAndOrder tries to order the selected services by contacting a car rental and, concurrently, a garage and a tow truck. It is defined as follows:

$$\begin{aligned}
\mathit{ChooseAndOrder} \triangleq\ & [x_{gps}]\,(\, p_{car} \bullet o_{choose}!\langle x_{list}\rangle \\
& \quad \mid [x_{garage}, x_{towTruck}, x_{rentalCar}] \\
& \qquad p_{car} \bullet o_{chosen}?\langle x_{garage}, x_{towTruck}, x_{rentalCar}\rangle. \\
& \qquad\quad (\, \mathit{OrderGarageAndTowTruck} \mid \mathit{RentCar}\,)\,)
\end{aligned}$$

$$\begin{aligned}
\mathit{OrderGarageAndTowTruck} \triangleq\ & [x_{garageInfo}] \\
& (\, x_{garage} \bullet o_{orderGar}!\langle p_{car}, x_{carData}\rangle \\
& \quad \mid p_{car} \bullet o_{garageFail}?\langle\rangle. \\
& \qquad (\, p_{car} \bullet o_{undo}!\langle cc\rangle \mid [p,o]\,(p \bullet o!\langle x_{loc}\rangle \mid p \bullet o?\langle x_{gps}\rangle)\,) \\
& \quad + p_{car} \bullet o_{garageOk}?\langle x_{gps}, x_{garageInfo}\rangle. \\
& \qquad (\, \mathit{OrderTowTruck} \\
& \qquad \mid p_{car} \bullet o_{undo}?\langle gar\rangle. \\
& \qquad\quad (\, x_{garage} \bullet o_{cancel}!\langle p_{car}\rangle \\
& \qquad\quad \mid p_{car} \bullet o_{undo}!\langle cc\rangle \mid p_{car} \bullet o_{undo}!\langle rc\rangle\,)\,)\,)
\end{aligned}$$

$$\begin{aligned}
\mathit{OrderTowTruck} \triangleq\ & [x_{towInfo}] \\
& (\, x_{towTruck} \bullet o_{orderTow}!\langle p_{car}, x_{loc}, x_{gps}\rangle \\
& \quad \mid p_{car} \bullet o_{towTruckFail}?\langle\rangle.\ p_{car} \bullet o_{undo}!\langle gar\rangle \\
& \quad + p_{car} \bullet o_{towTruckOK}?\langle x_{towInfo}\rangle\,)
\end{aligned}$$

$$\begin{aligned}
\mathit{RentCar} \triangleq\ & [x_{rcInfo}] \\
& (\, x_{rentalCar} \bullet o_{orderRC}!\langle p_{car}, x_{gps}\rangle \\
& \quad \mid p_{car} \bullet o_{rentalCarFail}?\langle\rangle.\ p_{car} \bullet o_{undo}!\langle cc\rangle \\
& \quad + p_{car} \bullet o_{rentalCarOK}?\langle x_{rcInfo}\rangle. \\
& \qquad p_{car} \bullet o_{undo}?\langle rc\rangle.\ x_{rentalCar} \bullet o_{redirect}!\langle p_{car}, x_{loc}\rangle\,)
\end{aligned}$$

If ordering a garage fails, the compensation of the credit card charge is invoked by sending a signal *cc* along the endpoint $p_{car} \bullet o_{undo}$, and the car's location (stored in x_{loc}) is assigned to variable x_{gps} (whose value will be passed to the rental car service). This assignment is rendered in COWS as a communication along the private endpoint $p \bullet o$. Otherwise, the tow truck ordering starts and the garage's location is assigned to variable x_{gps}. Moreover, a compensation handler is installed; it will be activated whenever tow truck ordering fails and, in that case, attempts to cancel the garage order (by invoking operation o_{cancel}) and to compensate the credit card charge and the rental car order (by sending signal *cc* and *rc* along the endpoint $p_{car} \bullet o_{undo}$). Renting a car proceeds concurrently and, in case of successful completion, the compensation handler for the redirection of the rented car is installed; otherwise, the compensation of the credit card charge is invoked.

The specification of the remaining components of the in-vehicle platform is as follows:

$$\mathit{SensorsMonitor} \triangleq p_{car} \bullet o_{engineFailure}!\langle ts, diagnosticData\rangle$$

$$\mathit{GpsSystem} \triangleq *\, p_{car} \bullet o_{reqLoc}?\langle\rangle.\ p_{car} \bullet o_{respLoc}!\langle gpsPos()\rangle$$

$$
\begin{aligned}
\mathit{Discovery} \triangleq\ & *[x_{gps}, x_{type}] \\
& p_{car} \bullet o_{findServ}?\langle x_{gps}, x_{type}\rangle. \\
& [p, o]\,(\, p \bullet o!\langle\rangle \mid p \bullet o?\langle\rangle.\ p_{car} \bullet o_{found}!\langle \mathit{servList}(x_{gps}, x_{type})\rangle \\
& \qquad\qquad + p \bullet o?\langle\rangle.\ p_{car} \bullet o_{notFound}!\langle\rangle\,)
\end{aligned}
$$

$$
\begin{aligned}
\mathit{Reasoner} \triangleq\ & *[x_{list}] \\
& p_{car} \bullet o_{choose}?\langle x_{list}\rangle. \\
& [p, o]\,(\, p \bullet o!\langle\rangle \\
& \qquad \mid p \bullet o?\langle\rangle.\ p_{car} \bullet o_{chosen}!\langle p_{garage\_1}, p_{towTruck\_1}, p_{rentalCar\_1}\rangle \\
& \qquad + p \bullet o?\langle\rangle.\ p_{car} \bullet o_{chosen}!\langle p_{garage\_2}, p_{towTruck\_1}, p_{rentalCar\_1}\rangle \\
& \qquad + \ldots + p \bullet o?\langle\rangle.\ p_{car} \bullet o_{chosen}!\langle p_{garage\_n}, p_{towTruck\_m}, p_{rentalCar\_r}\rangle\,)
\end{aligned}
$$

The COWS specification of the service *Bank* is composed of two persistent subservices: *BankInterface*, that is publicly invocable by customers, and *CreditRating*, that instead is an 'internal' service that can only interact with *BankInterface*. Specifically, *Bank* is the COWS term

$$[o_{check}, o_{checkOk}, o_{checkFail}]\,(\, *\,\mathit{BankInterface} \mid *\,\mathit{CreditRating}\,)$$

where *BankInterface* and *CreditRating* are defined as follows:

$$
\begin{aligned}
\mathit{BankInterface} \triangleq\ & [x_{cust}, x_{cc}, x_{amount}, x_{ts}] \\
& p_{bank} \bullet o_{charge}?\langle x_{cust}, x_{cc}, x_{amount}, x_{ts}\rangle. \\
& (\, p_{bank} \bullet o_{check}!\langle x_{ts}, x_{cc}, x_{amount}\rangle \\
& \quad \mid [x_{info}]\,(p_{bank} \bullet o_{checkFail}?\langle x_{ts}, x_{cc}, x_{info}\rangle.\ x_{cust} \bullet o_{resp}!\langle \mathit{fail}, x_{ts}, x_{info}\rangle \\
& \qquad + \ p_{bank} \bullet o_{checkOk}?\langle x_{ts}, x_{cc}, x_{info}\rangle. \\
& \qquad\quad [k']\,(\, x_{cust} \bullet o_{resp}!\langle ok, x_{ts}, x_{info}\rangle \\
& \qquad\qquad \mid p_{bank} \bullet o_{revoke}?\langle x_{ts}, x_{cc}\rangle.\ \mathbf{kill}(k')\,)\,)
\end{aligned}
$$

$$
\begin{aligned}
\mathit{CreditRating} \triangleq\ & [x_{ts}, x_{cc}, x_a] \\
& p_{bank} \bullet o_{check}?\langle x_{ts}, x_{cc}, x_a\rangle. \\
& [p, o]\,(\, p \bullet o!\langle\rangle \mid p \bullet o?\langle\rangle.\ p_{bank} \bullet o_{checkOk}!\langle x_{ts}, x_{cc}, \mathit{ratingInfo}(x_{cc}, x_a)\rangle \\
& \qquad\qquad + p \bullet o?\langle\rangle.\ p_{bank} \bullet o_{checkFail}!\langle x_{ts}, x_{cc}, \mathit{ratingInfo}(x_{cc}, x_a)\rangle\,)
\end{aligned}
$$

Whenever prompted by a client request, *BankInterface* creates an instance to serve that specific request and is immediately ready to concurrently serve other requests. Each instance forwards the request to *CreditRating*, by invoking the 'internal' operation o_{check} through the invoke activity $p_{bank} \bullet o_{check}!\langle x_{ts}, x_{cc}, x_{amount}\rangle$, then waits for a reply on one of the other two internal operations $o_{checkOK}$ and $o_{checkFail}$, by exploiting the receive-guarded choice operator, and finally sends the reply back to the client by means of a final invoke activity using the partner name of the client stored in the variable x_{cust}. In case of a positive answer, the possibility of revoking the request through invocation of operation o_{revoke} is enabled (in fact, should the discovery phase or ordering the services fail, the customer charge operation should be cancelled in order to implement the wanted transactional behaviour). Revocation causes deletion of the reply to the client, if this has still to be performed. Service *CreditRating* takes care of checking clients' requests and decides if they can be authorised or not. For the sake of simplicity, the choice between approving or not a request is left here completely non-deterministic.

OnRoadRepairServices is actually a composition of various on road services, i.e. it is

$$\mathit{Garage}_1 \mid \mathit{Garage}_2 \mid \mathit{TowTruck}_1 \mid \mathit{TowTruck}_2 \mid \mathit{RentalCar}_1 \mid \mathit{RentalCar}_2 \mid \ldots$$

Such concurrent on road services are all modelled in a similar way, e.g.

$$
\begin{aligned}
Garage_i \triangleq\ & *[x_{cust}, x_{sensorsData}, o_{checkOK}, o_{checkFail}] \\
& p_{garage\_i} \bullet o_{orderGar}?\langle x_{cust}, x_{sensorsData}\rangle. \\
& (\, p_{garage\_i} \bullet o_{checkOK}!\langle\rangle \mid p_{garage\_i} \bullet o_{checkFail}!\langle\rangle \\
& \mid p_{garage\_i} \bullet o_{checkFail}?\langle\rangle.\ x_{cust} \bullet o_{garageFail}!\langle\rangle \\
& + p_{garage\_i} \bullet o_{checkOK}?\langle\rangle. \\
& \qquad [k]\,(\, x_{cust} \bullet o_{garageOK}!\langle garageGPS_i, garageInfo_i\rangle \\
& \qquad\quad \mid p_{garage\_i} \bullet o_{cancel}?\langle x_{cust}\rangle.\ \mathbf{kill}(k)\,)\,)
\end{aligned}
$$

$$
\begin{aligned}
TowTruck_i \triangleq\ & *[x_{cust}, x_{carGps}, x_{garageGps}, o_{checkOK}, o_{checkFail}] \\
& p_{towTruck\_i} \bullet o_{orderTow}?\langle x_{cust}, x_{carGps}, x_{garageGps}\rangle. \\
& (\, p_{towTruck\_i} \bullet o_{checkOK}!\langle\rangle \mid p_{towTruck\_i} \bullet o_{checkFail}!\langle\rangle \\
& \mid p_{towTruck\_i} \bullet o_{checkFail}?\langle\rangle.\ x_{cust} \bullet o_{towTruckFail}!\langle\rangle \\
& + p_{towTruck\_i} \bullet o_{checkOK}?\langle\rangle.\ x_{cust} \bullet o_{towTruckOK}!\langle towTruckInfo_i\rangle\,)
\end{aligned}
$$

$$
\begin{aligned}
RentalCar_i \triangleq\ & *[x_{cust}, x_{gps}, o_{checkOK}, o_{checkFail}] \\
& p_{rentalCar\_i} \bullet o_{orderRC}?\langle x_{cust}, x_{gps}\rangle. \\
& (\, p_{rentalCar\_i} \bullet o_{checkOK}!\langle\rangle \mid p_{rentalCar\_i} \bullet o_{checkFail}!\langle\rangle \\
& \mid p_{rentalCar\_i} \bullet o_{checkFail}?\langle\rangle.\ x_{cust} \bullet o_{rentalCarFail}!\langle\rangle \\
& + p_{rentalCar\_i} \bullet o_{checkOK}?\langle\rangle. \\
& \qquad [k]\,(\, x_{cust} \bullet o_{rentalCarOK}!\langle rentalCarInfo_i\rangle \\
& \qquad\quad \mid [x_{newGps}]\, p_{rentalCar\_i} \bullet o_{redirect}?\langle x_{cust}, x_{newGps}\rangle.\ \mathbf{kill}(k)\,)\,)
\end{aligned}
$$

For simplicity, success or failure of orders are modelled by means of non-deterministic choice by exploiting internal operations $o_{checkOK}$ and $o_{checkFail}$.

References

[1] H. Garcia-Molina and K. Salem. Sagas. In *SIGMOD*, pages 249–259. ACM Press, 1987.

[2] N. Koch. Automotive case study: UML specification of on road assistance scenario, 2007. Sensoria report.

[3] P. Mayer, A. Schroeder, and N. Koch. A Model-Driven Approach to Service Orchestration. In *SCC*, volume 2, pages 533–536. IEEE Computer Society Press, 2008.

[4] OASIS WSBPEL TC. Web Services Business Process Execution Language Version 2.0. Technical report, OASIS, April 2007. Available at `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html`.

[5] SENSORIA. Software engineering for service-oriented overlay computers, 2005-2010. Web site: `http://www.sensoria-ist.eu/`.