Specification and Analysis of an Automotive Scenario

Technical Report

Authors: Alessandro Fantechi, Stefania Gnesi, Alessandro Lapadula, Franco Mazzanti,
Rosario Pugliese, and Francesco Tiezzi

Date: April 29, 2010
Institute: Dipartimento di Sistemi e Informatica, Università degli Studi di Firenze

1 Case study: an automotive scenario

In this section, we provide an informal description of the automotive case study, by also making use of UML-like diagrams, that will be formally specified and analysed in the next section.

The scenario is inspired to one of the case studies in the area of automotive systems defined and analysed within the EU project SENSORIA [2] and describes some functionalities that will be likely available in the near future. A brief description follows.

> 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. At this point, the *failure handler* installed in the in-vehicle computer system invokes an *assistance* service that, in its turn, contacts some garage, car rental and towing truck services, and tries to order them. To be authorised to order services, the assistance service has to deposit on behalf of the owner of the car a security payment, which will be given back if ordering the services fails.

A UML-like activity diagram of the assistance service using UML4SOA, an UML Profile for service-oriented systems [4], is shown in Figure 1. As usual, bars denote fork and join nodes, while diamonds denote decision and merge nodes. The assistance service is instantiated by a request from an in-vehicle computer system, received through the UML action SevereFailureAssistance, and consequently orchestrates the other services to reach its goal. The request is uniquely identified by the value of the input parameter id, which is subsequently used for correlation purposes. Then, the created instance invokes the *bank* to charge the driver's credit card with the security deposit amount. This is modelled by the action CardCharge for charging the credit card whose number is provided as an output parameter of the action call. If the credit card charge fails (because, e.g., there are not enough funds in the driver's bank account), the driver is informed by means of the FailureNotification action.

Services ordering is modelled by the UML actions OrderGarage, OrderTowTruck and RentCar. When the assistance service 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. If the order of the garage fails, the assistance service tries to make an appointment with the rental car service, by indicating the location of the stranded vehicle, where the car has to be handed over to the driver. Instead, if the order of the garage succeeds, the service concurrently tries to make an appointment with the rental service, by indicating the garage location as destination for the rental car, and with the towing service, providing the locations of the stranded vehicle and of the garage in order to tow the vehicle to the garage.

Besides interactions among services, the workflow described in Figure 1 also includes activities using concepts developed for long running business transactions (in e.g. [5]). These activities entail fault and compensation handling, sort of specific activities attempting to reverse the effects of previously committed activities, which are an important aspect of SOC applications. According to UML4SOA Profile, the installation of a compensation handler (represented by a dashed box) is modelled by a dashed edge labelled by the stereotype ≪compensation≫, and its activation by an activity labelled by ≪compensate≫. Specifically, in the considered scenario:

- the security deposit payment charged to the driver's credit card must be revoked if ordering the services completely fails, i.e. both garage/tow truck and car rental services reject the requests;
- the garage appointment has to be cancelled, if ordering a tow truck fails;
- the rental car delivery has to be redirected to the stranded car's actual location, if ordering a garage fails or a garage order cancellation is requested;
- instead, if ordering the car rental fails, it should not affect the tow truck and garage orders.

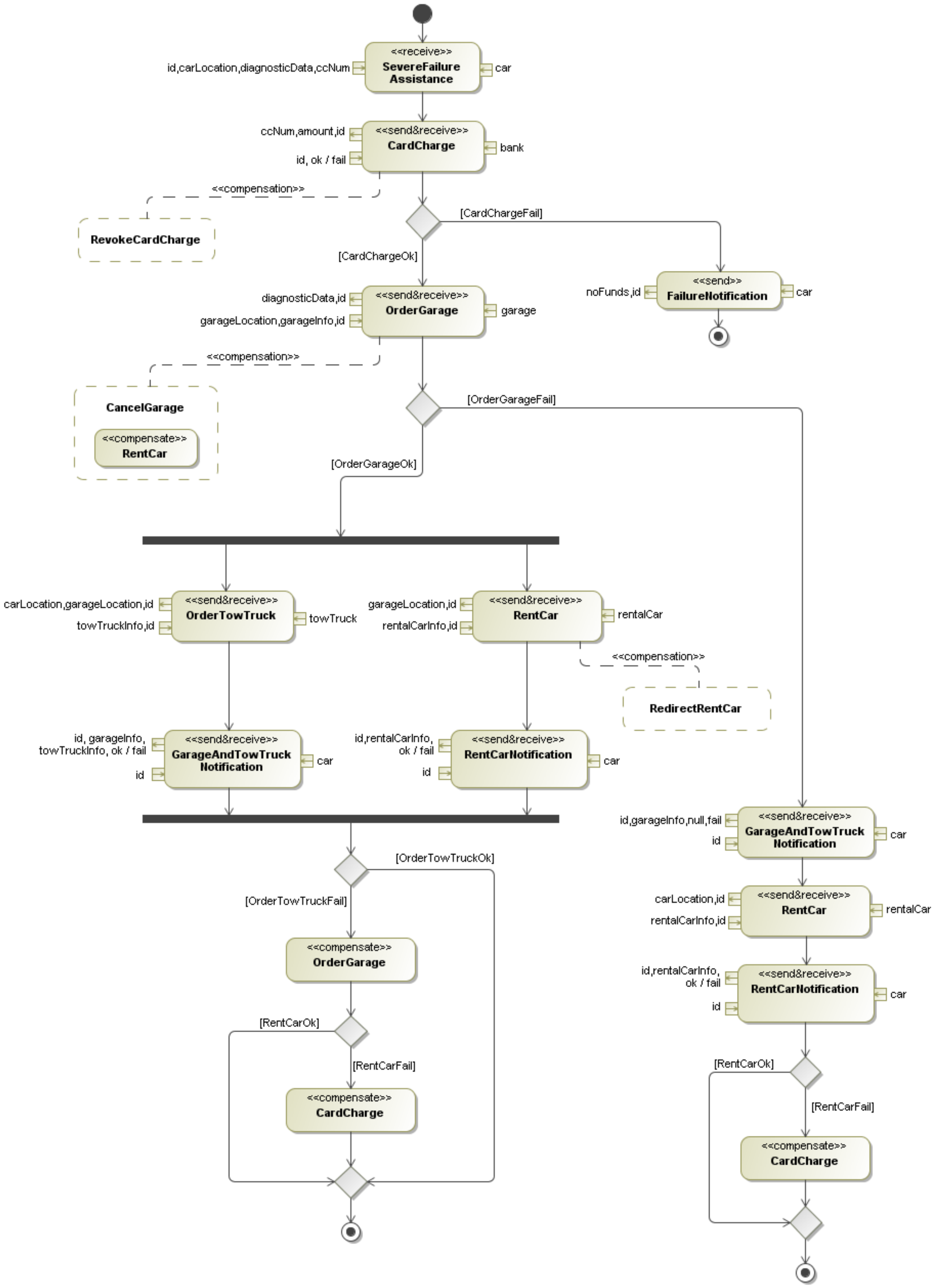

Figure 1: The assistance service

s	::=		(services)
		`nil`	(empty activity)
	|	`kill`(`k`)	(kill)
	|	`u.u'!` <*args*>	(invoke)
	|	`p.o?` <*params*> . s	(receive)
	|	$s_1 + \ldots + s_n$	(receive-guarded choice)
	|	$s_1 \mid s_2$	(parallel composition)
	|	$\{ s \}$	(protection)
	|	[`n`$\sharp$] s	(name delimitation)
	|	[`k`] s	(kill delimitation)
	|	[`X`] s	(variable delimitation)
	|	$* \, s$	(replication)
	|	`A`(*aparams*)	(call)
	|	`let A`(*fparams*) $= s_1 \ldots$ `B`(*fparams*) $= s_2$ `in` s' `end`	(let construct)
e	::=	`X` | `v` | $e_1 + e_2$ | e_1 `le` e_2 | ...	(expressions)
args	::=	e | *args*, *args*	(invoke arguments)
params	::=	`X` | `v`	(receive parameters)
	|	*params*, *params*	
fparams	::=	`X` | `n` | `k`	(formal parameters)
	|	*fparams*,*fparams*	
aparams	::=	`X` | `v` | `k`	(actual parameters)
	|	*aparams*, *aparams*	

Table 1: Syntax of COWS accepted by CMC

2 Formal specification and analysis of the scenario

We report in this section the complete COWS specification of the automotive scenario written in the syntax accepted by CMC. It is worth noticing that CMC requires that the abstraction rules to be applied to a specification be provided together with the specification itself. However, since in the sequel we report a few analyses relying on different abstractions of the automotive scenario, for the sake of presentation, we introduce each considered set of abstraction rules together with the set of SocL formulae that exploit them.

We first recall in Table 1 the syntax of COWS [3] accepted by CMC. *Killer labels* (ranged over by `k`, `k'`, ...) start with lower case letters and can only be used as argument of kill activities. *Variables* (ranged over by `X`, `Y`, ...) start with capital letters. *Service identifiers* (ranged over by `A`, `A'`, ...) start with capital letters and each of them has a fixed non-negative arity. *Names* (ranged over by `n`, `m`,...,`p`,`p'`,...,`o`,`o'`, ...) start with lower case letters. *Values* (ranged over by `v`, `v'`, ...) are integer numbers, booleans, or names. *Identifiers* (ranged over by `u`, `u'`, ...) are variables or names. The arguments of a receive-guarded choice must be receive activities. The expression operators + and `le` are defined as follows: if both e_1 and e_2 are evaluated as integer numbers then the evaluation of $e_1 + e_2$ returns the integer number corresponding to their sum, otherwise it returns the name corresponding to their concatenation; if both e_1 and e_2 are evaluated as values then the evaluation of e_1 `le` e_2 returns the boolean `true` if the value corresponding to e_1 is not greater than the value corresponding to e_2, otherwise it returns the boolean `false`.

The let construct permits to define services in a modular style, thus facilitating re-use of the same 'service code'. `let` A(*fparams*) =*s* ... `in` *s'* `end` behaves like *s'*, where calls to A can occur. A service *call* A(*aparams*) occurring in the body *s'* of a `let` A(*fparams*) =*s* ... `in` *s'* `end` behaves like the service obtained from *s* by replacing the formal parameters *fparams* with the corresponding actual parameters *aparams*.

The syntax of SocL accepted by CMC slightly differs from that presented in [1] mainly for what concern the notation to indicate correlation variables. In fact, given a variable `var`, its binding occurrence (i.e. $\underline{\texttt{var}}$ in SocL) is written `$var`, while its free occurrences are written `%var`.

2.1 Complete specification of the automotive scenario

The complete specification of the automotive scenario written in the syntax accepted by CMC is as follows.

```
let

-------------------------------------
--- Car's on-board computer system ---
-------------------------------------
 SensorsMonitor(car,diagnosticData) =
   car.engineFailure!<diagnosticData>


 GpsSystem(gps,car,carLocation) =
   * [Id] gps.reqLocation?<Id>. car.respLocation!<Id,carLocation>


 FailureHandler(gps,car,id,ccNum) =
   * [k][DiagnosticData]
     ( car.lowOilFailure?<DiagnosticData>. nil
--     + ...other failures...
       + car.engineFailure?<DiagnosticData>.
         ( -- Get car's current location
           gps.reqLocation!<id>
           | [CarLocation]
              car.respLocation?<id,CarLocation>.
                ( -- Invoke the assistance service
                  assistance.severeFailure!<car,id,CarLocation,DiagnosticData,ccNum>
                  |
                  -- Wait for answers
                    -- If the answer is a failure notification, stop the execution
                    [Failure] car.failureNotification?<Failure,id>. kill(k)
                  | -- Otherwise, collect the data
                    [GarageAndTowTruckResponse][GarageInfo][TowTruckInfo]
                    car.garageAndTowTruckNotification?<GarageAndTowTruckResponse,GarageInfo,TowTruckInfo,id>.
                       assistance.garageAndTowTruckNotificationAck!<id>
                    |
                    [RentalCarResponse][RentalCarInfo]
                    car.rentCarNotification?<RentalCarResponse,RentalCarInfo,id>.
                       ( assistance.rentCarNotificationAck!<id>
                         |
                         -- Wait for a possible rental car redirection
                         car.rentalCarRedirected?<id>. nil
                       )
                )
         )
     )


 Car(car,id,gps,diagnosticData,carLocation,ccNum)=
   ( SensorsMonitor(car,diagnosticData)
     | GpsSystem(gps,car,carLocation)
     | FailureHandler(gps,car,id,ccNum) )



--------------------
--- Bank service ---
--------------------
```

```
BankInterface(check,checkOk,checkFail) =
  * [Cust][Cc][Amount][Id]
    bank.charge?<Cust,Cc,Amount,Id>.
      ( bank.check!<Id,Cc,Amount>
        | bank.checkFail?<Id>. Cust.chargeFail!<Id>
          + bank.checkOk?<Id>.
              [k] ( Cust.chargeOK!<Id> | bank.revoke?<Id>.kill(k) ) )


CreditRating(check,checkOk,checkFail) =
  * [Id] [Cc] [A]
    bank.check?<Id,Cc,A>.
      [p#][o#] (p.o!<> | p.o?<>. bank.checkOk!<Id>
                       + p.o?<>. bank.checkFail!<Id>)


Bank = [check#] [checkOk#] [checkFail#]
      ( BankInterface(check,checkOk,checkFail)
        | CreditRating(check,checkOk,checkFail) )



--------------------------
--- Assistance service ---
--------------------------
 TowTruckOrdering(towTruck,Car,Id,CarLocation,GarageLocation,GarageInfo) =
 -- Tow truck ordering
 [TowTruckInfo]
 ( towTruck.orderTowTruck!<assistance,CarLocation,GarageLocation,Id>
   | assistance.orderTowTruckFail?<TowTruckInfo,Id>.
         ( -- Notify the driver that the garage/tow truck order failed
           Car.garageAndTowTruckNotification!<fail,GarageInfo,TowTruckInfo,Id>
           |
           assistance.garageAndTowTruckNotificationAck?<Id>.
              ( assistance.orderTowTruckFail!<Id> | assistance.end!<Id> )
         )
     +
     assistance.orderTowTruckOK?<TowTruckInfo,Id>.
         ( -- Notify the driver that the garage/tow truck order succeeded
           Car.garageAndTowTruckNotification!<ok,GarageInfo,TowTruckInfo,Id>
           |
           assistance.garageAndTowTruckNotificationAck?<Id>. assistance.end!<Id>
         )
 )



 RentalCarOrdering(rentalCar,Car,Id,PrimaryLocation,SecondaryLocation) =
 -- Rental car ordering
 [RentalCarInfo]
 ( rentalCar.rentCar!<assistance,PrimaryLocation,Id>
   | assistance.rentCarFail?<RentalCarInfo,Id>.
         ( -- Notify the driver that the rental car order failed
           Car.rentCarNotification!<fail,RentalCarInfo,Id>
           |
           assistance.rentCarNotificationAck?<Id>.
              ( assistance.orderRentCarFail!<Id> | assistance.end!<Id> )
         )
     +
     assistance.rentCarOK?<RentalCarInfo,Id>.
         ( -- Install the RentCar compensation handler
           assistance.undo?<rentCar,Id>.
             ( -- Redirect the rental car
               rentalCar.redirect!<SecondaryLocation,Id>
               | Car.rentalCarRedirected!<Id>
             )
           |
           -- Notify the driver that the rental car order succeeded
           Car.rentCarNotification!<ok,RentalCarInfo,Id>
           |
           assistance.rentCarNotificationAck?<Id>. assistance.end!<Id>
         )
 )


 Ordering(Car,Id,CarLocation,DiagnosticData,garage,towTruck,rentalCar) =
   -- Garage ordering
   [GarageLocation][GarageInfo]
   ( garage.orderGarage!<assistance,DiagnosticData,Id>
     | assistance.orderGarageFail?<GarageInfo,Id>.
          ( -- Notify the driver that the garage order failed
            Car.garageAndTowTruckNotification!<fail,GarageInfo,null,Id>
            |
            assistance.garageAndTowTruckNotificationAck?<Id>.
              ( -- Rent a car directed to the car's current location
                RentalCarOrdering(rentalCar,Car,Id,CarLocation,CarLocation)
                |
                -- if the rental car order failed, compensate the card charge
                assistance.orderRentCarFail?<Id>.
                  assistance.undo!<cardCharge,Id>
              )
          )
       +
       assistance.orderGarageOK?<GarageLocation,GarageInfo,Id>.
          ( -- Install the OrderGarage compensation handler
            assistance.undo?<orderGarage,Id>.
              ( -- Cancel the garage order
                garage.cancel!<Id>
                | -- Redirect the rental car
                  assistance.undo!<rentCar,Id>
              )
            |
            -- Order the tow truck and the rental car (in parallel)
            TowTruckOrdering(towTruck,Car,Id,CarLocation,GarageLocation,GarageInfo)
            |
            RentalCarOrdering(rentalCar,Car,Id,GarageLocation,CarLocation)
            |
            assistance.end?<Id>.
              assistance.end?<Id>.
                -- Both orders are terminated:
                -- if the tow truck order failed, compensate the garage order
                assistance.orderTowTruckFail?<Id>.
                  ( assistance.undo!<orderGarage,Id>
                    | -- if also the rental car order failed,
                      -- compensate the card charge
                      assistance.orderRentCarFail?<Id>.
                        assistance.undo!<cardCharge,Id>
                  )
          )
   )


 Assistance(garage,towTruck,rentalCar) =
   * [Car][Id][CarLocation][DiagnosticData][CcNum]
     assistance.severeFailure?<Car,Id,CarLocation,DiagnosticData,CcNum>.
       ( -- Try to charge the driver's credit card
         bank.charge!<assistance,CcNum,amount,Id>
         | assistance.chargeFail?<Id>.
             Car.failureNotification!<noFunds,Id>
           + assistance.chargeOK?<Id>.
             ( -- Start the ordering phase
               Ordering(Car,Id,CarLocation,DiagnosticData,garage,towTruck,rentalCar)
               |
               -- Install the CardCharge compensation handler
               assistance.undo?<cardCharge,Id>. bank.revoke!<Id>
             )
       )
```

```
--- On-road assistance service ---
----------------------------------
 Garage(garage,garageLocation,garageInfo,garageFailureInfo) =
   * [Cust][Data][Id]
     garage.orderGarage?<Cust,Data,Id>.
       ( garage.checkOK!<Id> | garage.checkFAIL!<Id>
         | garage.checkFAIL?<Id>. Cust.orderGarageFail!<garageFailureInfo,Id>
           + garage.checkOK?<Id>.
               [k] ( Cust.orderGarageOK!<garageLocation,garageInfo,Id>
                     |
                     garage.cancel?<Id>. kill(k) ) )


 TowTruck(towTruck,towTruckInfo,towTruckFailureInfo) =
   * [Cust][CarLocation][GarageLocation][Id]
     towTruck.orderTowTruck?<Cust,CarLocation,GarageLocation,Id>.
       ( towTruck.checkOK!<Id> | towTruck.checkFAIL!<Id>
         | towTruck.checkFAIL?<Id>. Cust.orderTowTruckFail!<towTruckFailureInfo,Id>
           + towTruck.checkOK?<Id>. Cust.orderTowTruckOK!<towTruckInfo,Id> )


 RentalCar(rentalCar,rentalCarInfo,rentalCarFailureInfo) =
   * [Cust][Location][Id]
     rentalCar.rentCar?<Cust,Location,Id>.
       ( rentalCar.checkOK!<Id> | rentalCar.checkFAIL!<Id>
         | rentalCar.checkFAIL?<Id>. Cust.rentCarFail!<rentalCarFailureInfo,Id>
           + rentalCar.checkOK?<Id>.
               [k] ( Cust.rentCarOK!<rentalCarInfo,Id>
                     | [NewLocation]
                       rentalCar.redirect?<NewLocation,Id>. kill(k) ) )



in

-----------------------------------------
--- Automotive scenario specification ---
-----------------------------------------
   Car(car1,id1,gpsCar1,diagnosticData1,carLocation1,ccNum1)
   | Car(car2,id2,gpsCar2,diagnosticData2,carLocation2,ccNum2)
   |
   Assistance(garage1,towTruck2,rentalCar1)
   |
   Bank()
   |
   Garage(garage1,garageLocation1,garageInfo1,garageFailureInfo1)
   | Garage(garage2,garageLocation2,garageInfo2,garageFailureInfo2)
   |
   TowTruck(towTruck1,towTruckInfo1,towTruckFailureInfo1)
   | TowTruck(towTruck2,towTruckInfo2,towTruckFailureInfo2)
   |
   RentalCar(rentalCar1,rentalCarInfo1,rentalCarFailureInfo1)
   | RentalCar(rentalCar2,rentalCarInfo2,rentalCarFailureInfo2)

end
```

2.2 Verification of the abstract properties presented in the Introduction

The abstraction rules used for this analysis are the following.

```
Abstractions {
 Action assistance.severeFailure<*,$id,*,*,*>  -> request(road_assistance,$id)
 Action *.garageAndTowTruckNotification<ok,*,*,$id>      -> responseOk(road_assistance,$id)
 Action *.rentCarNotification<ok,*,$id>    -> responseOk(road_assistance,$id)
 Action *.failureNotification<*,$id>     -> responseFail(road_assistance,$id)
 Action *.garageAndTowTruckNotification<fail,*,*,$id>      -> responseFail(road_assistance,$id)
 Action *.rentCarNotification<fail,*,$id>    -> responseFail(road_assistance,$id)
 State  assistance.severeFailure? -> accepting_request(road_assistance)
}
```

The SocL formulae written in the syntax accepted by CMC are as follows.

```
1) -- Available service --
   AG(accepting_request(road_assistance))


2) -- Parallel service --
   AG [request(road_assistance,$var)]
      E[true {not (responseOk(road_assistance,%var) or
                   responseFail(road_assistance,%var))}
         U accepting_request(road_assistance)]


3) -- Sequential service --
   AG [request(road_assistance,$var)]
      A[not accepting_request(road_assistance) {true}
         U {responseOk(road_assistance,%var) or
            responseFail(road_assistance,%var)} true]


4) -- One-shot service --
   AG [request(road_assistance,$var)]
      AG not accepting_request(road_assistance)


5) -- Off-line service --
   AG [request(road_assistance,$var)]
      AF {responseFail(road_assistance,%var)} true



6) -- Cancelable service --
   AG [request(road_assistance,$var)]
      A[accepting_cancel(road_assistance,%var) {true}
         W {responseOk(road_assistance,%var) or
            responseFail(road_assistance,%var)} true]


7) -- Revocable service --
   EF {responseOk(road_assistance,$var)}
      EF (accepting_undo(road_assistance,%var))


8) -- Responsive service --
   AG [request(road_assistance,$var)]
      AF {responseOk(road_assistance,%var) or
          responseFail(road_assistance,%var)} true


9) -- Single-response service --
   AG [request(road_assistance,$var)]
      not EF {responseOk(road_assistance,%var) or
              responseFail(road_assistance,%var)}
             EF {responseOk(road_assistance,%var) or
                 responseFail(road_assistance,%var)} true


10) -- Multiple-response service --
    AG [request(road_assistance,$var)]
       AF {responseOk(road_assistance,%var) or
           responseFail(road_assistance,%var)}
       AF {responseOk(road_assistance,%var) or
           responseFail(road_assistance,%var)} true


11) -- No-response service --
    AG [request(road_assistance,$var)]
       not EF {responseOk(road_assistance,%var) or
               responseFail(road_assistance,%var)} true


12) -- Reliable service --
```

```
    AG [request(road_assistance,$var)]
       AF {responseOk(road_assistance,%var)} true
```

2.3 Verification of some request-response properties

The abstraction rules used for this analysis are the following.

```
Abstractions {
 Action assistance.severeFailure<*,$id,*,*,*>         -> request(road_assistance,$id)
 Action *.garageAndTowTruckNotification<ok,*,*,$id>   -> responseOk(road_assistance,$id,truckGarage)
 Action *.rentCarNotification<ok,*,$id>               -> responseOk(road_assistance,$id,rentalCar)
 Action *.failureNotification<*,$id>                  -> responseFail(road_assistance,$id,truckGarage)
 Action *.failureNotification<*,$id>                  -> responseFail(road_assistance,$id,rentalCar)
 Action *.garageAndTowTruckNotification<fail,*,*,$id> -> responseFail(road_assistance,$id,truckGarage)
 Action *.rentCarNotification<fail,*,$id>             -> responseFail(road_assistance,$id,rentalCar)
}
```

The SocL formulae written in the syntax accepted by CMC are as follows.

```
(F1) AG [request(road_assistance,$var)]
        AF {responseOk(road_assistance,%var,truckGarage) or
            responseFail(road_assistance,%var,truckGarage)} true


(F2) AG [request(road_assistance,$var)]
        AF {responseOk(road_assistance,%var,rentalCar) or
            responseFail(road_assistance,%var,rentalCar)} true


(F3) AG [responseOk(road_assistance,$var,$order)]
        not EF {responseFail(road_assistance,%var,%order)} true


(F4) AG [responseFail(road_assistance,$var,$order)]
        not EF {responseOk(road_assistance,%var,%order)} true
```

2.4 Analysis of other services of the automotive scenario

The abstraction rules used for this analysis are the following.

```
Abstractions {
 Action gpsCar1.reqLocation<$id>    -> request(gps1,$id)
 Action *.respLocation<$id,*> -> responseOk(gps1,$id)
 State  gpsCar1.reqLocation?        -> accepting_request(gps1)

 Action bank.charge<*,*,*,$id>      -> request(charge,$id)
 Action *.chargeOK<$id>             -> responseOk(charge,$id)
 Action *.chargeFail<$id>           -> responseFail(charge,$id)
 Action bank.revoke<$id>            -> undo(charge,$id)
 State  bank.charge?                -> accepting_request(charge)
 State  bank.revoke?<$id>           -> accepting_undo(charge,$id)

 Action rentalCar1.rentCar<*,*,$id> -> request(rental_car1,$id)
 Action *.rentCarOK<*,$id>          -> responseOk(rental_car1,$id)
 Action *.rentCarFail<*,$id>        -> responseFail(rental_car1,$id)
 State  rentalCar1.rentCar?         -> accepting_request(rental_car1)
}
```

The SocL formulae written in the syntax accepted by CMC are as follows.

```
(F5) AG(accepting_request(gps1))


(F6) AG [request(gps1,$var)] AF {responseOk(gps1,$var)} true


(F7) AG(accepting_request(charge))


(F8) AG [request(charge,$id)]
        AF {responseOk(charge,%id) or responseFail(charge,%id)}
           not EF {responseOk(charge,%id) or responseFail(charge,%id)} true

(F9) AG [responseOk(charge,$id)]
        A[ accepting_undo(charge,%id) {true} W {undo(charge,%id)} true]


(F10) AG(accepting_request(rental_car1))


(F11) AG [request(rental_car1,$customer)]
         AF {responseOk(rental_car1,%customer) or
             responseFail(rental_car1,%customer)}
         not EF {responseOk(rental_car1,%customer) or
                 responseFail(rental_car1,%customer)} true
```

2.5 Verification of orchestration and compensation properties

The abstraction rules used for this analysis are the following.

```
Abstractions {
 Action assistance.severeFailure<*,$id,*,*,*>         -> request(road_assistance,$id)
 Action *.garageAndTowTruckNotification<ok,*,*,$id>   -> responseOk(road_assistance,$id,truckGarage)
 Action *.rentCarNotification<ok,*,$id>               -> responseOk(road_assistance,$id,rentalCar)
 Action *.failureNotification<*,$id>                  -> responseFail(road_assistance,$id,truckGarage)
 Action *.failureNotification<*,$id>                  -> responseFail(road_assistance,$id,rentalCar)
 Action *.garageAndTowTruckNotification<fail,*,*,$id> -> responseFail(road_assistance,$id,truckGarage)
 Action *.rentCarNotification<fail,*,$id>             -> responseFail(road_assistance,$id,rentalCar)

 Action bank.charge<*,*,*,$id>     -> request(charge,$id)
 Action *.chargeOK<$id>            -> responseOk(charge,$id)
 Action *.chargeFail<$id>          -> responseFail(charge,$id)
 Action bank.revoke<$id>           -> undo(charge,$id)

 Action *.orderGarageOK<*,*,$id>   -> responseOk(garage,$id)
 Action *.cancel<$id>              -> undo(garage,$id)
 Action *.orderTowTruckFail<*,$id> -> responseFail(towtruck,$id)
}
```

The SocL formulae written in the syntax accepted by CMC are as follows.

```
(F12) AG [responseOk(charge,$id)]
         AF {responseOk(road_assistance,%id,rentalCar) or
             responseOk(road_assistance,%id,truckGarage) or
             undo(charge,%id)} true


(F13) not EF {responseOk(charge,$id)}
            EF {responseOk(road_assistance,%id,rentalCar) or
                responseOk(road_assistance,%id,truckGarage)}
               EF {undo(charge,%id)} true


(F14) not EF {responseOk(charge,$id)}
             EF {undo(charge,%id)}
```

```
                EF {responseOk(road_assistance,%id,rentalCar) or
                    responseOk(road_assistance,%id,truckGarage)} true


(F15) AG [responseOk(garage,$var)]
         AG [responseFail(towtruck,%var)]
            AF {undo(garage,%var)} true
```

References

[1] A. Fantechi, S. Gnesi, A. Lapadula, F. Mazzanti, R. Pugliese, and F. Tiezzi. A Logical Verification Methodology for Service-Oriented Computing, 2010. Submitted paper. An extended abstract appeared in *FASE*, LNCS 4961, pages 230-245, 2008, Springer.

[2] N. Koch. Automotive Case Study: UML Specification of On Road Assistance Scenario. Sensoria report, 2007.
`http://rap.dsi.unifi.it/sensoria/files/FAST_report_1_2007_ACS_UML.pdf`.

[3] A. Lapadula, R. Pugliese, and F. Tiezzi. A calculus for orchestration of web services. Technical report, DSI, Università di Firenze, 2008. `http://rap.dsi.unifi.it/cows/papers/cows-esop07-full.pdf`. An extended abstract appeared in *ESOP*, LNCS 4421, pages 33-47, 2007, Springer.

[4] P. Mayer, A. Schroeder, and N. Koch. Mdd4soa: Model-driven service orchestration. In *EDOC*, pages 203–212. IEEE Computer Society Press, 2008.

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