Sensoria

016004

Software Engineering for Service-Oriented Overlay Computers

www.sensoria-ist.eu

Manual for Using the UMC Model of the Automotive Case Study

Author(s): Maurice H. ter Beek and Franco Mazzanti (ISTI–CNR)

Date of preparation: March 12, 2010
Revision: final
Dissemination level: PU

Contract start date: September 1, 2005 Duration: 48 months
Project coordinator: Martin Wirsing (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, CIR

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

Information Society
Technologies

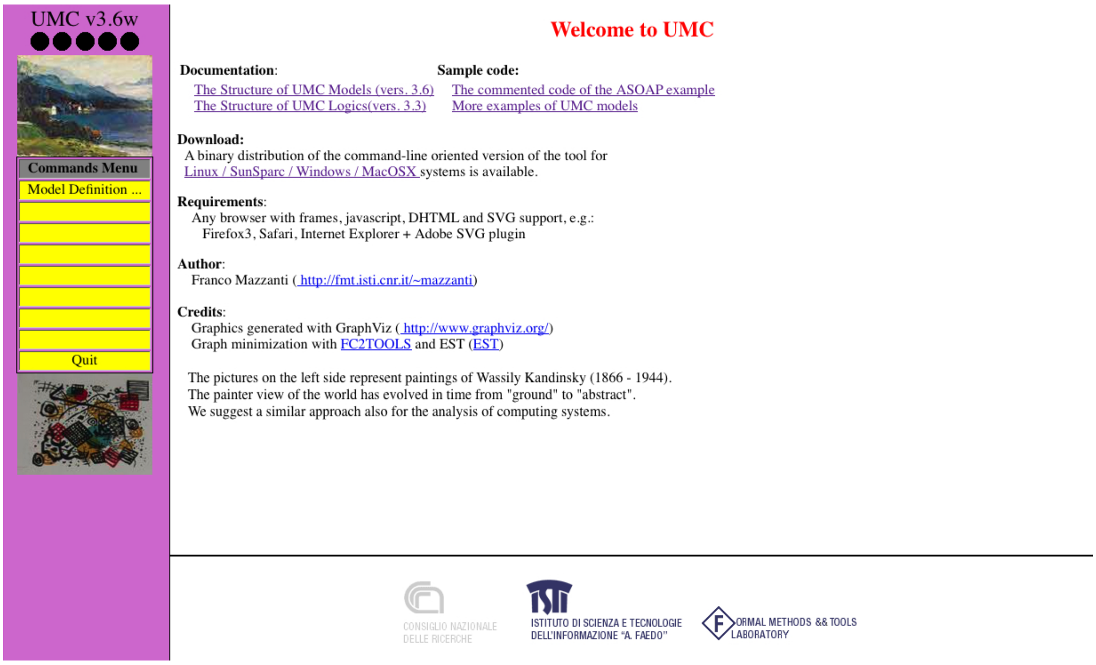
Executive Summary

We show how to use a UMC model of the on road assistance scenario of SENSORIA's Automotive case study, described in [2], and verify properties formulated in the service-oriented temporal logic SocL.

Contents

1 UMC

UMC [1, 4] is an on-the-fly model checker (its current prototype can be experimented via a web interface [6], which also includes a user guide [5]). UMC allows the efficient verification of SocL formulae over a set of communicating UML state machines. SocL [4] is an event- and state-based, branching-time, efficiently verifiable, parametric temporal logic that was specifically designed to capture peculiar aspects of services. UMC's web interface is depicted in Fig. 1.


UMC v3.6w
●●●●●
Commands Menu
Model Definition ...
Quit
CONSIGLIO NAZIONALE DELLE RICERCHE
ISTITUTO DI SCIENZA E TECNOLOGIE DELL'INFORMAZIONE "A. FAEDO"
FORMAL METHODS && TOOLS LABORATORY

Welcome to UMC

Documentation:
The Structure of UMC Models (vers. 3.6)
The Structure of UMC Logics(vers. 3.3)

Sample code:
The commented code of the ASOAP example
More examples of UMC models

Download:
A binary distribution of the command-line oriented version of the tool for Linux / SunSparc / Windows / MacOSX systems is available.

Requirements:
Any browser with frames, javascript, DHTML and SVG support, e.g.:
Firefox3, Safari, Internet Explorer + Adobe SVG plugin

Author:
Franco Mazzanti (http://fmt.isti.cnr.it/~mazzanti)

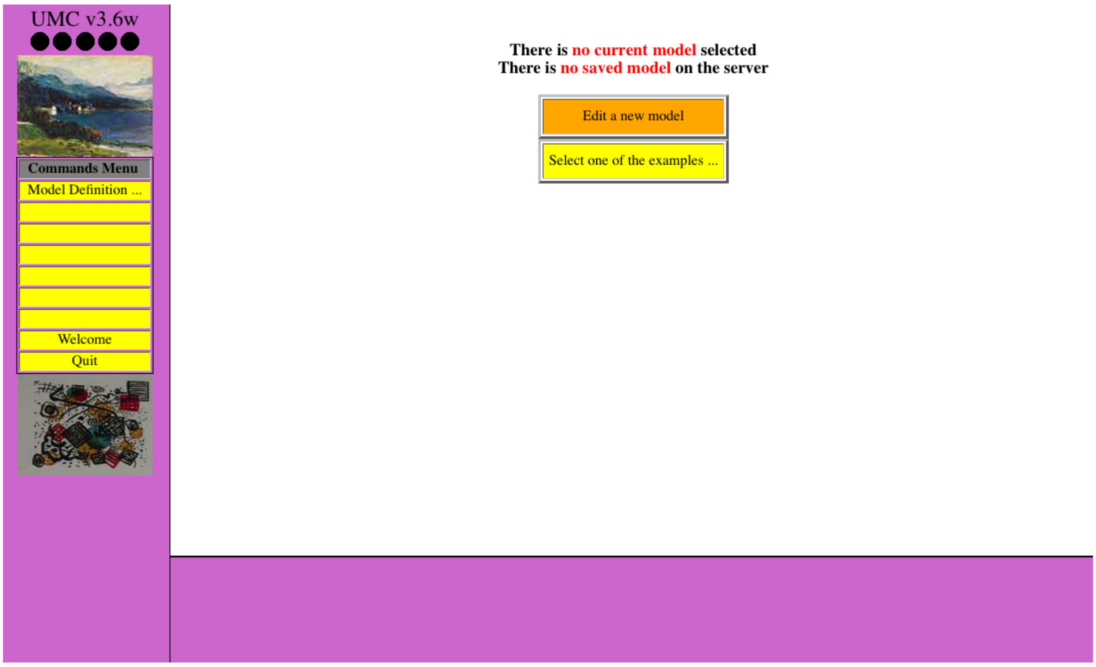
Credits:
Graphics generated with GraphViz (http://www.graphviz.org/)
Graph minimization with FC2TOOLS and EST (EST)

The pictures on the left side represent paintings of Wassily Kandinsky (1866 - 1944).
The painter view of the world has evolved in time from "ground" to "abstract".
We suggest a similar approach also for the analysis of computing systems.

Figure 1: UMC web interface.

1.1 Selecting a UMC model

By selecting "Model Definition . . . " in the Commands Menu on the left, one obtains Fig. 2.



There is no current model selected
There is no saved model on the server

Figure 2: Selecting a model.

Subsequently clicking “Select one of the examples . . . ” brings one to Fig. 3 (it might be necessary to use the scrollbar to select the UMC model `00-automotive.umc`).

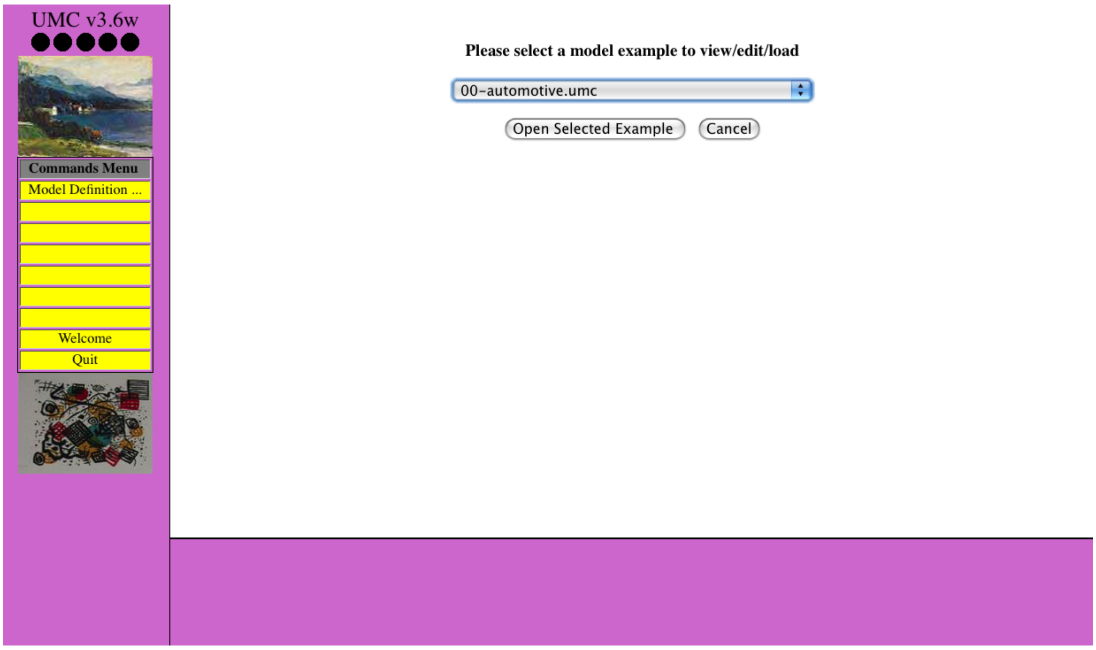
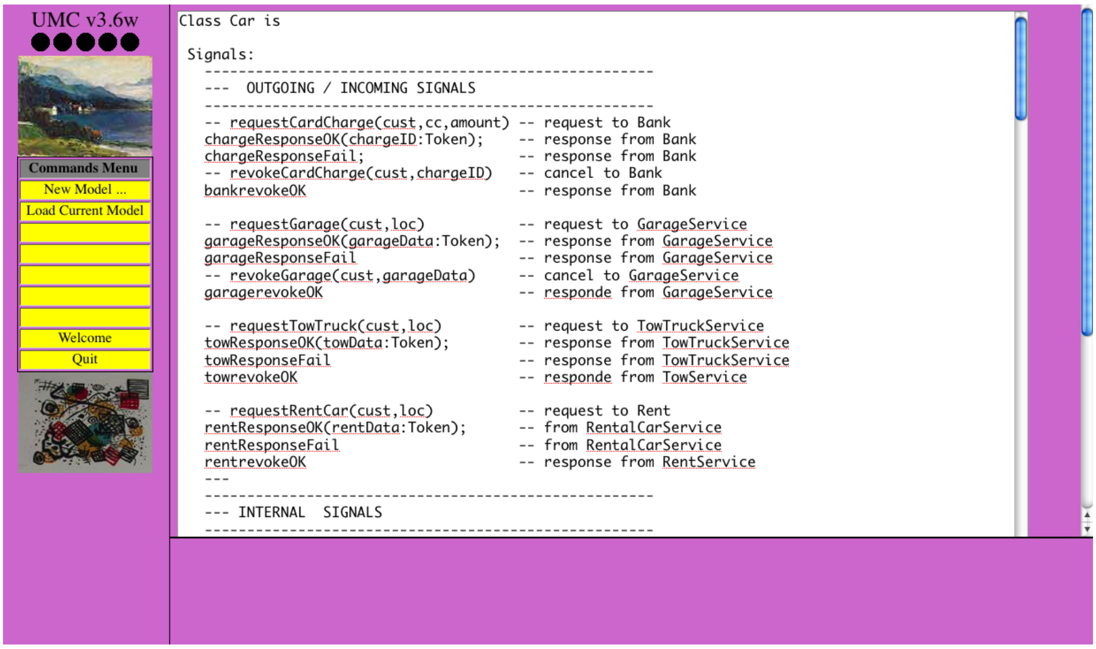


Figure 3: Selecting an example model.

Clicking “Open Selected Example” leads to Fig. 4: the UMC model of the On road assistance scenario of the Automotive case study.



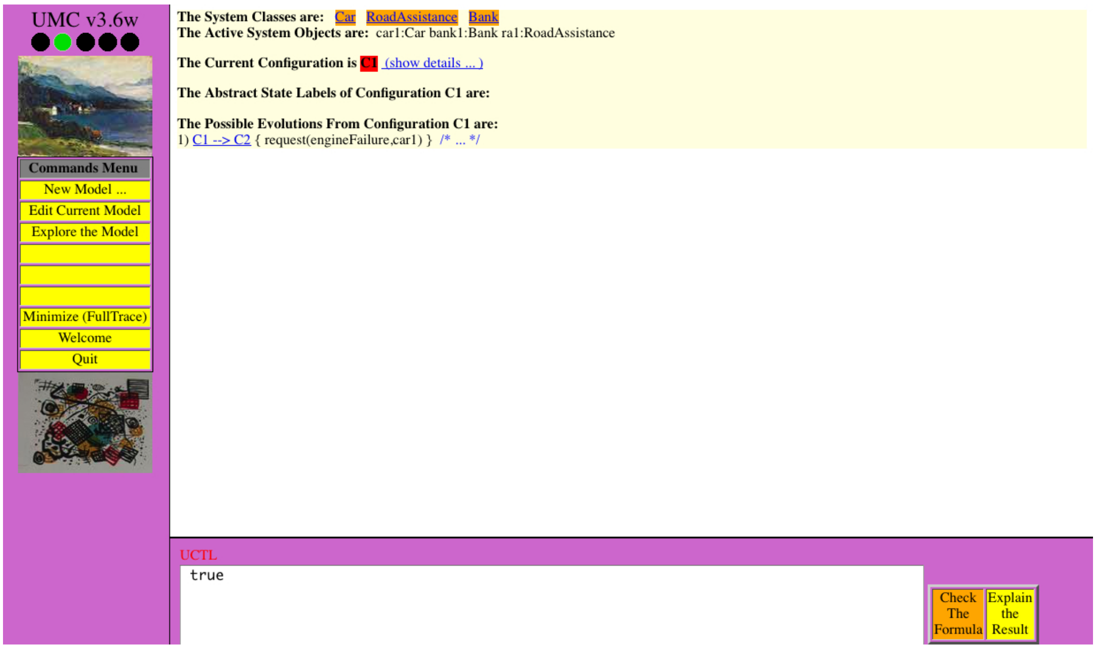
```
Class Car is

 Signals:
   -----------------------------------------------------
   ---  OUTGOING / INCOMING SIGNALS
   -----------------------------------------------------
   -- requestCardCharge(cust,cc,amount) -- request to Bank
   chargeResponseOK(chargeID:Token);    -- response from Bank
   chargeResponseFail;                  -- response from Bank
   -- revokeCardCharge(cust,chargeID)   -- cancel to Bank
   bankrevokeOK                         -- response from Bank

   -- requestGarage(cust,loc)           -- request to GarageService
   garageResponseOK(garageData:Token);  -- response from GarageService
   garageResponseFail                   -- response from GarageService
   -- revokeGarage(cust,garageData)     -- cancel to GarageService
   garagerevokeOK                       -- responde from GarageService

   -- requestTowTruck(cust,loc)         -- request to TowTruckService
   towResponseOK(towData:Token);        -- response from TowTruckService
   towResponseFail                      -- response from TowTruckService
   towrevokeOK                          -- responde from TowService

   -- requestRentCar(cust,loc)          -- request to Rent
   rentResponseOK(rentData:Token);      -- from RentalCarService
   rentResponseFail                     -- from RentalCarService
   rentrevokeOK                         -- response from RentService
   ---
   -----------------------------------------------------
   --- INTERNAL  SIGNALS
   -----------------------------------------------------
```

Figure 4: UMC model of On road assistance.

Using the scrollbar on the right, one can inspect this UMC code.

1.2 Experimenting with a UMC model

To start experimenting the UMC model, one must select “Load Current Model” in the Commands Menu on the left, resulting in Fig. 5. This figure shows the model’s classes and active objects, as well as its current (initial) configuration.



The System Classes are: Car RoadAssistance Bank
The Active System Objects are: car1:Car bank1:Bank ra1:RoadAssistance

The Current Configuration is C1 (show details ...)

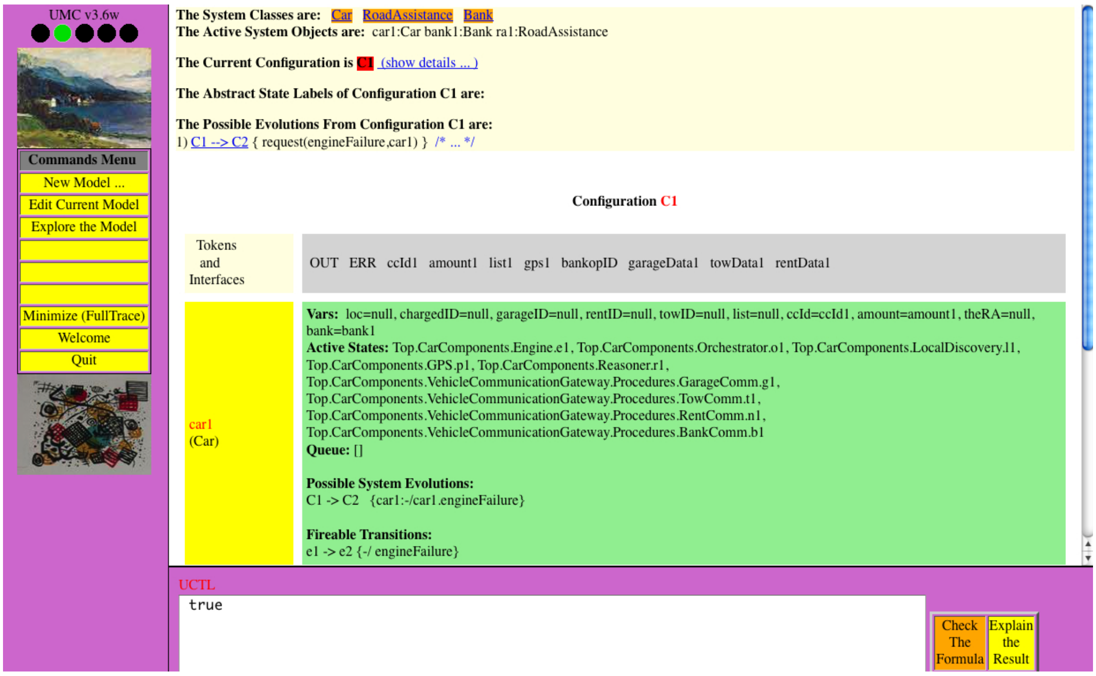
The Abstract State Labels of Configuration C1 are:

The Possible Evolutions From Configuration C1 are:
1) C1 --> C2 { request(engineFailure,car1) } /* ... */

```
true
```

Figure 5: A loaded UMC model.

The latter can be inspected further by clicking “(show details . . .)”, which results in Fig. 6 (using the scrollbar on the right details of variables, active states, event queues and possible evolutions of all active objects can be consulted).



The System Classes are: Car RoadAssistance Bank
The Active System Objects are: car1:Car bank1:Bank ra1:RoadAssistance

The Current Configuration is C1 (show details ...)

The Abstract State Labels of Configuration C1 are:

The Possible Evolutions From Configuration C1 are:
1) C1 --> C2 { request(engineFailure,car1) } /* ... */

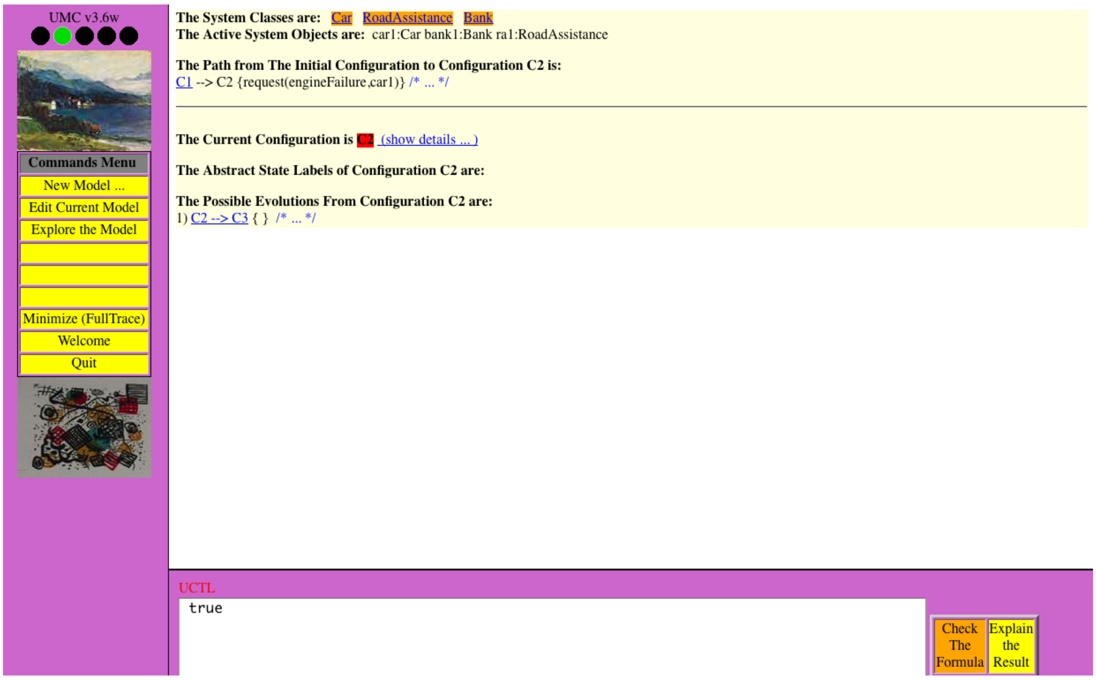
Configuration C1

Tokens and Interfaces	OUT ERR ccId1 amount1 list1 gps1 bankopID garageData1 towData1 rentData1
car1 (Car)	**Vars:** loc=null, chargedID=null, garageID=null, rentID=null, towID=null, list=null, ccId=ccId1, amount=amount1, theRA=null, bank=bank1 **Active States:** Top.CarComponents.Engine.e1, Top.CarComponents.Orchestrator.o1, Top.CarComponents.LocalDiscovery.l1, Top.CarComponents.GPS.p1, Top.CarComponents.Reasoner.r1, Top.CarComponents.VehicleCommunicationGateway.Procedures.GarageComm.g1, Top.CarComponents.VehicleCommunicationGateway.Procedures.TowComm.t1, Top.CarComponents.VehicleCommunicationGateway.Procedures.RentComm.n1, Top.CarComponents.VehicleCommunicationGateway.Procedures.BankComm.b1 **Queue:** [] **Possible System Evolutions:** C1 -> C2 {car1:-/car1.engineFailure} **Fireable Transitions:** e1 -> e2 {-/ engineFailure}

```
true
```

Figure 6: A model’s details.

Another possibility provided in Fig. 5 is to perform an evolution step by clicking on “C1 −−> C2”, which results in Fig. 7 (which can be ‘repeated’ in the obvious way to perform more evolution steps).



The System Classes are: Car RoadAssistance Bank
The Active System Objects are: car1:Car bank1:Bank ra1:RoadAssistance

The Path from The Initial Configuration to Configuration C2 is:
C1 --> C2 {request(engineFailure,car1)} /* ... */

The Current Configuration is C2 (show details ...)

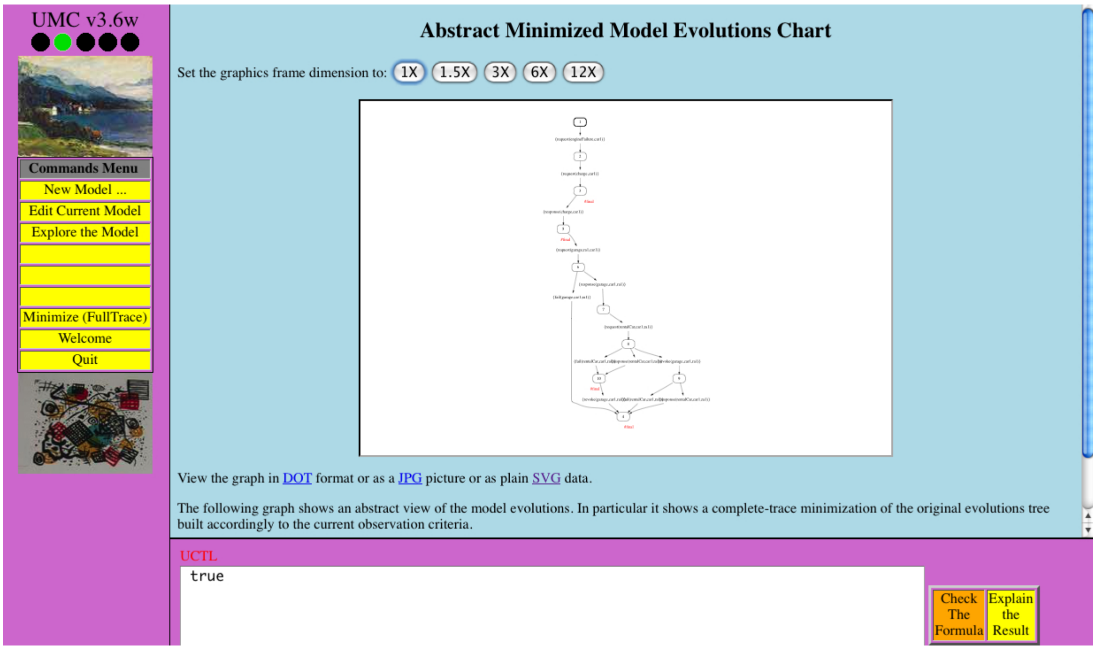
The Abstract State Labels of Configuration C2 are:

The Possible Evolutions From Configuration C2 are:
1) C2 --> C3 { } /* ... */

```
true
```

Figure 7: An evolution step.

Yet another possibility in Fig. 5 is to create a minimized abstract evolution graph of the model by clicking on “Minimize (FullTrace)”, which results in Fig. 8.



Abstract Minimized Model Evolutions Chart

Set the graphics frame dimension to: 1X 1.5X 3X 6X 12X

View the graph in DOT format or as a JPG picture or as plain SVG data.

The following graph shows an abstract view of the model evolutions. In particular it shows a complete-trace minimization of the original evolutions tree built accordingly to the current observation criteria.

```
true
```

Figure 8: A minimized abstract evolution graph.

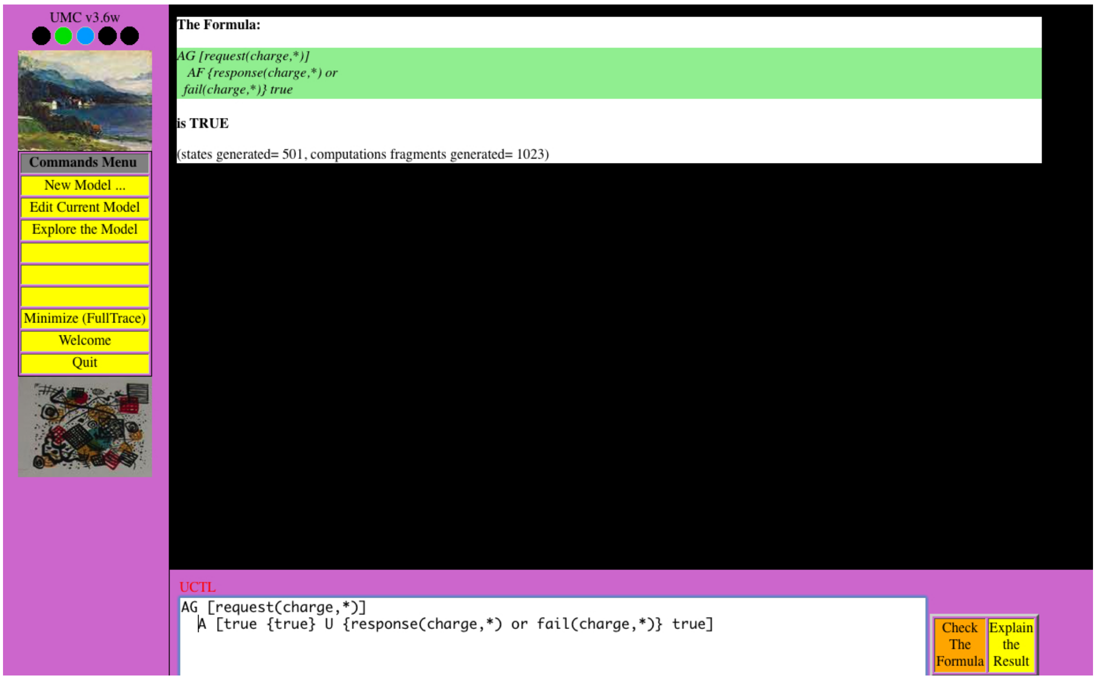
2 Verification with UMC

Two examples show how to use UMC to verify SocL formulae over the model of Sect. 1 (described in detail in [2]). More properties that we have verified, inspired by the Patterns of service properties listed in [4], can be found in [2]. As a first example, we verify that the `Bank` service is *responsive*, i.e. it guarantees a response to each received request. To this aim, it suffices to verify the SocL formula

$$\mathtt{AG}\,[\mathtt{request}(\mathtt{charge},*)]\;\mathtt{A}\,[\mathtt{true}\;\{\mathtt{true}\}\;\mathtt{U}\;\{\mathtt{response}(\mathtt{charge},*)\;\mathtt{or}\;\mathtt{fail}(\mathtt{charge},*)\}\;\mathtt{true}],$$

which states that each time action requestCardCharge takes place, always at a certain moment action chargeResponseOK or chargeResponseFail takes place.[1] More intuitively: If the Car requests the Bank to charge a credit card, then the Bank will surely reply with a notification of either a successful or a failed attempt to charge the credit card.

Verifying the above formula can be done by inserting it in the field labelled UCTL (of which SocL is a specialized version) on the lower side, which by default contains the formula true, and subsequently pushing the button “Check The Formula” on the lower right side. This results in Fig. 9, i.e. the above formula is TRUE. Note that a UMC model needs to be loaded before verifying properties, so the UCTL field only appears from Fig. 5 onward.



The Formula:

AG [request(charge,)]*
AF {response(charge,) or*
fail(charge,)} true*

is TRUE

(states generated= 501, computations fragments generated= 1023)

```
AG [request(charge,*)]
  A [true {true} U {response(charge,*) or fail(charge,*)} true]
```

Figure 9: Result (true) of a verification.

2.1 Interpreting a Counterexample

As a second example, we now repeat the above operations to verify whether the Garage service is *reliable*, i.e. whether it guarantees a *successful* response whenever it accepts a request (for this service). To this aim, it suffices to verify the SocL formula

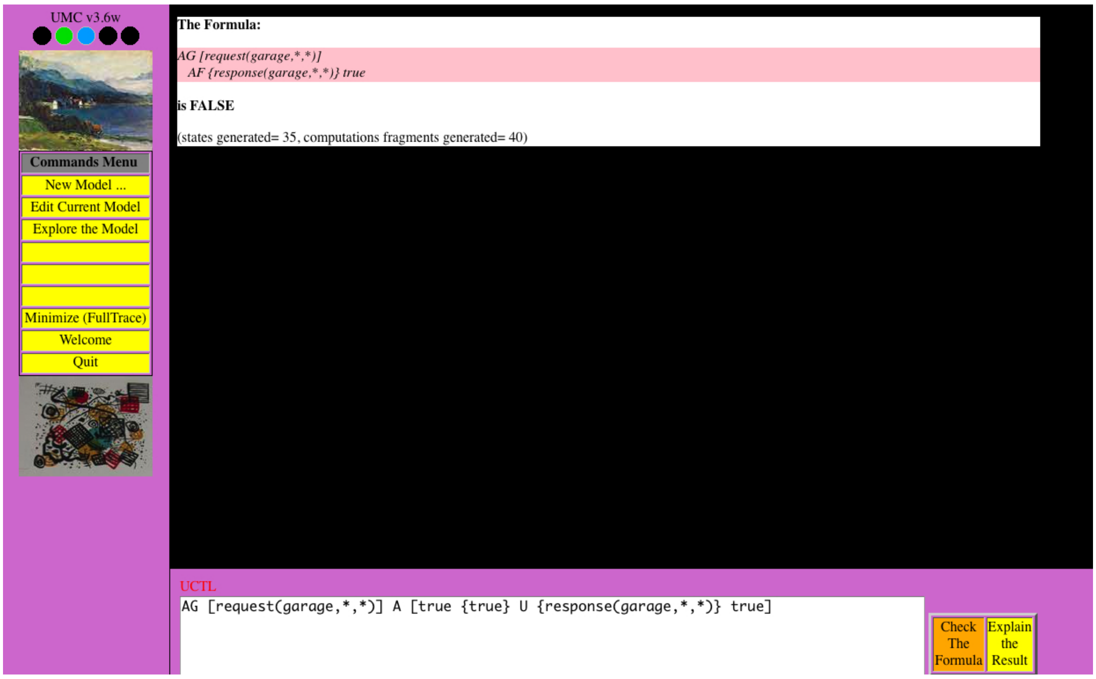
$$\mathtt{AG}\,[\mathtt{request}(\mathtt{garage},*,*)]\;\mathtt{A}\,[\mathtt{true}\;\{\mathtt{true}\}\;\mathtt{U}\;\{\mathtt{response}(\mathtt{garage},*,*)\}\;\mathtt{true}],$$

which states that each time action requestGarage takes place, always at a certain moment action garageResponseOK takes place.[2] More intuitively: Reservation requests from Car to Garage are always followed by a notification of success.

Doing so results in Fig. 10, i.e. this formula is FALSE. Note that this is not surprising: The Garage service might be temporarily unable to provide the requested service (so it sends the unsuccessful response garageResponseFail). Note that the Garage service is responsive, i.e. a formula similar to the formula verified earlier for the Bank service does hold also for the Garage service.

[1] Note that for the sake of readability we abbreviated the actions we used in the formulae in [2]: requestCardCharge = request(charge,*), chargeResponseOK = response(charge,*) and chargeResponseFail = fail(charge,*)

[2] Again, note that for the sake of readability we abbreviated the actions used in the formulae in [2]: requestGarage = request(garage,*,*) and garageResponseOK = response(garage,*,*)

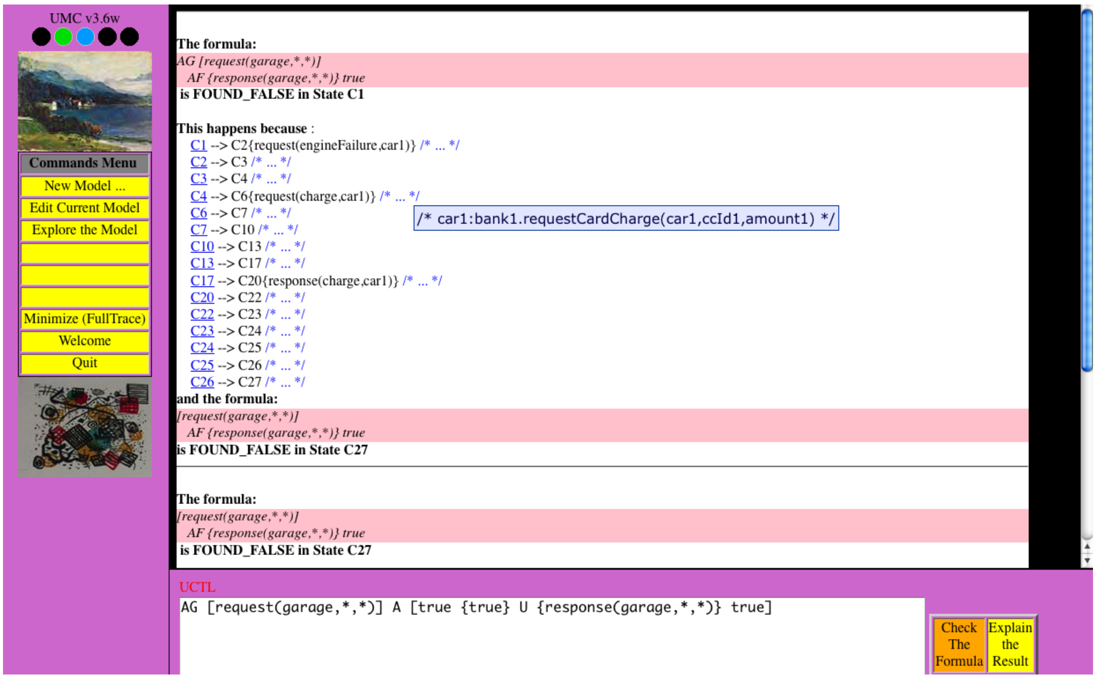


```
The Formula:
AG [request(garage,*,*)]
  AF {response(garage,*,*)} true
is FALSE
(states generated= 35, computations fragments generated= 40)
```

```
AG [request(garage,*,*)] A [true {true} U {response(garage,*,*)} true]
```

Figure 10: Result (false) of a verification.

Pushing the button "Explain the Result" on the lower right side results in Fig. 11, displaying the counterexample produced by UMC. The node names are hyperlinks which, when followed, allow one to observe all details of that configuration. Furthermore, while abstract transition labels are always fully displayed on the right-hand side of the transitions, their corresponding underlying ground events (which are useful for understanding what exactly is happening in the ground model's evolutions) are shown as dynamic tooltips that appear when the cursor is moved over the "/*...*/" regions. Note that the explanation returned by UMC has the form of a (partial) proof, in the sense that not only the *witnessing* model fragment but also the subformulae holding in the various substates, are put in evidence; moreover, only what are considered the *useful* parts of the explanation are shown.



```
The formula:
AG [request(garage,*,*)]
  AF {response(garage,*,*)} true
 is FOUND_FALSE in State C1

This happens because :
  C1 --> C2{request(engineFailure,car1)} /* ... */
  C2 --> C3 /* ... */
  C3 --> C4 /* ... */
  C4 --> C6{request(charge,car1)} /* ... */
  C6 --> C7 /* ... */          /* car1:bank1.requestCardCharge(car1,ccId1,amount1) */
  C7 --> C10 /* ... */
  C10 --> C13 /* ... */
  C13 --> C17 /* ... */
  C17 --> C20{response(charge,car1)} /* ... */
  C20 --> C22 /* ... */
  C22 --> C23 /* ... */
  C23 --> C24 /* ... */
  C24 --> C25 /* ... */
  C25 --> C26 /* ... */
  C26 --> C27 /* ... */
and the formula:
[request(garage,*,*)]
  AF {response(garage,*,*)} true
is FOUND_FALSE in State C27

The formula:
[request(garage,*,*)]
  AF {response(garage,*,*)} true
 is FOUND_FALSE in State C27
```

```
AG [request(garage,*,*)] A [true {true} U {response(garage,*,*)} true]
```

Figure 11: Counterexample of a formula.

References

[1] M.H. ter Beek, A. Fantechi, S. Gnesi and F. Mazzanti, A state/event-based model-checking approach for the analysis of abstract system properties. In *Science of Computer Programming*, 2010.

[2] M.H. ter Beek, S. Gnesi, N. Koch and F. Mazzanti, Formal Verification of an Automotive Scenario in Service-Oriented Computing. In *Proceedings of the 30th International Conference on Software Engineering (ICSE'08), Leipzig, Germany*, ACM Press, New York, 2008, 613–622.

[3] M.H. ter Beek, F. Mazzanti, and S. Gnesi, CMC–UMC: A Framework for the Verification of Abstract Service-Oriented Properties. In *Proceedings of the 24th Annual ACM Symposium on Applied Computing (SAC'09), Honolulu, Hawaii*, USA, ACM Press, New York, 2009, 2111–2117.

[4] S. Gnesi and F. Mazzanti, An Abstract, on the Fly Framework for the Verification of Service Oriented Systems. In [7], 2010.

[5] F. Mazzanti, Designing UML models with UMC. Technical Report 2009-TR-43, ISTI–CNR, 2009.

[6] UMC: `http://fmt.isti.cnr.it/umc/`.

[7] M. Wirsing and M. Hölzl (Eds.), *Rigorous Software Engineering for Service-Oriented Systems—Results of the SENSORIA project on Software Engineering for Service-Oriented Computing*, Springer, 2010.

A The Full UMC Model of the On Road Assistance Scenario

We append the full UMC model used in this paper and in [2]. It is listed among the example models on the UMC web interface [6] as `00-automotive.umc`.[3]

Once loaded, this UMC model consists of 501 states. Furthermore, its system classes are `Car`, `Bank`, and `RoadAssistance`, while its active system objects are `car1: Car`, `bank1: Bank`, and `ra1: RoadAssistance`.

UMC allows its users to specify an *observation mode* of the system under analysis, in which one explicitly specifies a set of *hiding* and *renaming* rules that precisely define the structural information or events one is interested to observe, possibly reshaping them to fit a standard format, hiding the rest (cf. [1] for more details). This is done by adding to the model's UMC encoding an `Abstractions` section containing this list of abstraction rules. The abstractions that are relevant for the fomulae verified in Sect. 2 are as follows:

```
Abstractions {
  ...
  Action: $1:requestCardCharge -> request(charge,$1)
  Action: $1.chargeResponseOK -> response(charge,$1)
  Action: $1.chargeResponseFail -> fail(charge,$1)
  Action: $1.requestGarage($2,$3) -> request(garage,$1,$2)
  Action: $1:$2.garageResponseOK -> response(garage,$2,$1)
  Action: $1:$2.garageResponseFail -> fail(garage,$2,$1)
  ...
}
```

[3] In [3] we used a different UMC model of the On road assistance scenario of the Automotive case study, which is instead listed among the example models on the UMC web interface [6] as `00-automotive-SAC09.umc`

```
Class Car is

 Signals:
   -----------------------------------------------------
   --- OUTGOING / INCOMING SIGNALS
   -----------------------------------------------------
   -- requestCardCharge(cust,cc,amount) -- request to Bank
   chargeResponseOK(chargeID:Token);    -- response from Bank
   chargeResponseFail;                  -- response from Bank
   -- revokeCardCharge(cust,chargeID)   -- cancel to Bank
   bankrevokeOK                         -- response from Bank
   --
   -- requestGarage(cust,loc)           -- request to GarageService
   garageResponseOK(garageData:Token);  -- response from GarageService
   garageResponseFail                   -- response from GarageService
   -- revokeGarage(cust,garageData)     -- cancel to GarageService
   garagerevokeOK                       -- responde from GarageService
   --
   -- requestTowTruck(cust,loc)         -- request to TowTruckService
   towResponseOK(towData:Token);        -- response from TowTruckService
   towResponseFail                      -- response from TowTruckService
   towrevokeOK                          -- responde from TowService
   --
   -- requestRentCar(cust,loc)          -- request to Rent
   rentResponseOK(rentData:Token);      -- from RentalCarService
   rentResponseFail                     -- from RentalCarService
   rentrevokeOK                         -- response from RentService
   -----------------------------------------------------
   --- INTERNAL SIGNALS
   -----------------------------------------------------
   engineFailure;             -- Engine -> Orchestrator
   --
   reqLoc;                    -- Orchestrator -> GPS
   respLoc(mygps:Token);      -- GPS -> Orchestrator
   --
   findServ(mygps:Token);     -- Orchestrator -> LocalDiscovery
   found(mylist:Token);       -- LocalDiscovery -> Orchestrator
   notFound;                  -- LocalDiscovery -> Orchestrator
   --
   choose;                     -- Orchestrator -> Reasoner
   chosen(myRA:RoadAssistance) -- Reasoner -> Orchestrator
   --
   bankcharge           -- Orchestrator -> VehicleCommunicationGateway
   bankOK               -- VehicleCommunicationGateway -> Orchestrator
   bankFail             -- VehicleCommunicationGateway -> Orchestrator
   bankrevoke           -- Orchestrator -> VehicleCommunicationGateway
   --
   orderGarage          -- Orchestrator -> VehicleCommunicationGateway
   garageOK             -- VehicleCommunicationGateway -> Orchestrator
   garageFail           -- VehicleCommunicationGateway -> Orchestrator
   garagerevoke         -- Orchestrator -> VehicleCommunicationGateway
   --
   orderTowTruck        -- Orchestrator -> VehicleCommunicationGateway
   towOK                -- VehicleCommunicationGateway -> Orchestrator
   towFail              -- VehicleCommunicationGateway -> Orchestrator
   towrevoke            -- unused
   --
   rentCar              -- Orchestrator -> VehicleCommunicationGateway
   failedRentCar        -- VehicleCommunicationGateway -> Orchestrator
   carRented            -- VehicleCommunicationGateway -> Orchestrator
   rentrevoke           -- Orchestrator -> VehicleCommunicationGateway

 Vars:
   loc: Token := null;  -- used by Orchestrator
   chargedID: Token;
   garageID: Token;
   rentID: Token;
   towID: Token;
   list: Token := null;
   ccId: Token := ccId1;
   amount: Token := amount1;
   theRA: RoadAssistance;
```

```
  bank: Bank := bank1;

State Top =
 CarComponents(
     Engine[ e1, e2] ,
     Orchestrator[
        o1,
        EnablingPhase(
           CardCharge[o2, o3, o4, final] ,
           FindServices[o6, o7, o8, o9, final]),
        ServiceSelection,
        OrderServices(
           o11,
           o12,
           o13,
           TowAndCar(
              OrderTow[
                 o14,
                 o15,
                 CompensateAll(
                    CompensateBank[x1, x4],
                    CompensateGarage[x2, x5],
                    CompensateRent[x3, x6]),
                 final]  ,
              OrderCar[o17, o18, o19, final]
              )),
        final] ,
     LocalDiscovery[l1] ,
     GPS[p1] ,
     Reasoner[r1] ,
     VehicleCommunicationGateway[
        Procedures(
           GarageComm [g1, g2, g3, g4, g5, g6],
           TowComm [t1, t2, t3,t4, t5, t6],
           RentComm [n1, n2, n3, n4, n5, n6],
           BankComm [b1, b2, b3, b4, b5, b6]
        )]
     )

State RentComm Defers rentrevoke
State BankComm Defers bankrevoke

Transitions:
-- Engine
  e1 -> e2 {- / engineFailure}
-- Orchestrator
  o1 -> EnablingPhase {engineFailure}
  --- CardCharge
  o2 -> o3 {- / self.bankcharge}    -- activate bank calling procedure
  o3 -> o4 {bankFail}
  o3 -> CardCharge.final {bankOK}
  --- FindServices
  o6 -> o7 {- / self.reqLoc}                      -- call GPS
  o7 -> o8 {respLoc(mygps) /                      -- response from GPS
             loc := mygps; self.findServ(mygps)}  -- call LocalDiscoveryService
  o8 -> o9 {notFound / bankrevoke}                -- FAILURE with bank compensation
  o8 -> FindServices.final {found(mylist:Token)}          -- respond from local discovery
  EnablingPhase -> ServiceSelection {- / self.choose}     -- activate reasoner
  ServiceSelection -> OrderServices
           {chosen(myRA) /                        -- response from reasoner
              theRA := myRA}
  --- OrderServices
  o11 -> o12 {- / self.orderGarage}               -- activate garagecomm
  o12 -> o13 {garageFail / self.bankrevoke}       -- FAILURE with bank compensation
  o12 -> TowAndCar {garageOK}
  ---  OrderTow
  o14 -> o15 {- / self.orderTowTruck}  -- activare towcomm
  o15 -> OrderTow.final {towOK}
  o15 -> CompensateAll  {towFail}      -- FAILURE with bank and garage and rent compensation
  -- CompensateAll
  x1 -> x4 {- / self.bankrevoke}
  x2 -> x5 {- / self.garagerevoke}
```

```
   x3 -> x6 {- / self.rentrevoke}
   --- OrderCar
   o17 -> o18 {- / self.rentCar}
   o18 -> OrderCar.final {carRented}
   o18 -> o19 {failedRentCar}
   OrderServices -> Orchestrator.final
 -- LocalDiscovery
   l1 -> l1 {findServ(mygps) / self.found(list1)}   -- uses loc not modelled
   l1 -> l1 {findServ(mygps) / self.notFound}
 -- Reasoner
   r1 -> r1 {choose / self.chosen(ra1)}
 -- GPS
   p1 -> p1 {reqLoc / self.respLoc(gps1)}
 -- GarageProcedures
   g1 -> g2 {orderGarage / theRA.requestGarage(self,loc)} -- call external garage service
   g2 -> g3 {garageResponseOK(garageData) / garageID := garageData; self.garageOK}
   g2 -> g4 {garageResponseFail / self.garageFail}        -- response Fail
   -- compensations
   g1 -> g6 {garagerevoke}
   g3 -> g5 {garagerevoke / theRA.revokeGarage(self, garageID)}  -- cancel external request
   g5 -> g6 {garagerevokeOK}                                     -- response from service
   g4 -> g6 {garagerevoke}
 -- TowProcedures
   t1 -> t2 {orderTowTruck / theRA.requestTowTruck(self,loc)}  -- call external garage service
   t2 -> t3 {towResponseOK(towData) / towID := towData; self.towOK}   -- response OK
   t2 -> t4 {towResponseFail / self.towFail}                          -- response Fail
   --
   t1 -> t6 {towrevoke}
   t3 -> t5 {towrevoke / theRA.revokeTowTruck(self,towID)}       -- cancel external request
   t5 -> t6 {towrevokeOK}                                        -- response from service
   t4 -> t6 {towrevoke}
 -- RentProcedures
   n1 -> n2 {rentCar / theRA.requestRentCar(self,loc)}       -- call external rental service
   n2 -> n3 {rentResponseOK(rentData) / rentID := rentData;  self.carRented}  -- response OK
   n2 -> n4 {rentResponseFail/ self.failedRentCar}           -- response Fail
   -- compensations
   n1 -> n6 {rentrevoke}
   n3 -> n5 {rentrevoke / theRA.revokeRentCar(self,rentID)}     -- cancel external request
   n5 -> n6 {rentrevokeOK}                                      -- response from service
   n4 -> n6 {rentrevoke}
 -- BankProcedures
   b1 -> b2 {bankcharge / bank.requestCardCharge(self, ccId, amount)} -- call external service
   b2 -> b3 {chargeResponseOK(chargeID) / chargedID := chargeID; self.bankOK} -- response OK
   b2 -> b4 {chargeResponseFail / self.bankFail}                              -- response Fail
   -- compensations
   b1 -> b6 {bankrevoke}
   b3 -> b5 {bankrevoke/ bank.revokeCardCharge(self,chargedID)}  -- cancel external service
   b5 -> b6 {bankrevokeOK}                                       -- response from service
   b4 -> b6 {bankrevoke}

end Car

Class Bank is

 Signals:
     requestCardCharge(cust:Car, cc:Token, amount:Token);
     -- replies:  cust.chargeResponseOK(chargeID)
     --                 cust.chargeResponseFail
     revokeCardCharge(cust:Car, chargeID:Token);
     --  replies:   bankrevokeOK

 State Top = s1

 Transitions:
   s1 -> s1 { requestCardCharge(cust,cc,amount) / cust.chargeResponseOK(bankopID) }
   s1 -> s1 { requestCardCharge(cust,cc,amount) / cust.chargeResponseFail }
   s1 -> s1 { revokeCardCharge(cust,chargeID) / cust.bankrevokeOK }

end Bank

Class RoadAssistance is
```

```
 Signals:
   ------- GARAGE SERVICES -------
   requestGarage(cust:Car,loc:Token);
    -- replies:  garageResponseOK(garageData) to car
    --           garageResponseFail           to car
    --
    revokeGarage(cust:Car,garageData:Token);
    -- replies:  garagerevokeOK
   -------- TOWTRUCK SERVICES -------
   requestTowTruck(cust:Car,loc:Token);
    -- replies:  towResponseOK(towData) to car
    --           towResponseFail        to car
    --
    revokeTowTruck(cust:Car, towData:Token)
    -- replies:  cust.towrevokeOK
   ------- RENTAL SERVICES -------
   requestRentCar(cust:Car,loc:Token);
    -- replies:  rentResponseOK(rentData) to car
    --           rentResponseFail         to car
    --
    revokeRentCar(cust:Car, rentData:Token)
    -- replies:  cust.rentrevokeOK

 State Top = Services
 State Services = GarageService / TowTruckService / RentalCarService
 State GarageService = g1
 State TowTruckService = t1
 State RentalCarService = r1

 Transitions:

   --  garage services
   g1 -> g1 { requestGarage(cust,loc) / cust.garageResponseOK(garageData1) }
   g1 -> g1 { requestGarage(cust,loc) / cust.garageResponseFail }
   g1 -> g1 { revokeGarage(cust,garageData) / cust.garagerevokeOK }
   -- tow truck
   t1 -> t1 { requestTowTruck(cust,loc) / cust.towResponseOK(towData1) }
   t1 -> t1 { requestTowTruck(cust,loc) / cust.towResponseFail }
   t1 -> t1 { revokeTowTruck(cust,towData) / cust.towrevokeOK }
   -- rental
   r1 -> r1 { requestRentCar(cust,loc) / cust.rentResponseOK(rentData1) }
   r1 -> r1 { requestRentCar(cust,loc) / cust.rentResponseFail }
   r1 -> r1 { revokeRentCar(cust,rentData) / cust.rentrevokeOK }

end RoadAssistance

----------------
Objects:
----------------
bankopID, rentData1, garageData1, towData1, ccId1, amount1, gps1, list1: Token;

car1: Car;
bank1: Bank;
ra1: RoadAssistance


Abstractions {
 Action: $1:engineFailure ->  request(engineFailure,$1)
 Action: $1:requestCardCharge -> request(charge,$1)
 Action: $1.chargeResponseOK -> response(charge,$1)
 Action: $1.chargeResponseFail -> fail(charge,$1)
 Action: $1.requestGarage($2,$3) -> request(garage,$1,$2)
 Action: $1:$2.garageResponseOK -> response(garage,$2,$1)
 Action: $1:$2.garageResponseFail -> fail(garage,$2,$1)
 Action: $1:$2.revokeGarage -> revoke(garage,$1,$2)
 Action: $1:$2.requestRentCar -> request(rentalCar,$1,$2)
 Action: $1:$2.rentResponseOK-> response(rentalCar,$2,$1)
 Action: $1:$2.rentResponseFail-> fail(rentalCar,$2,$1)
 State: inState(car1.Orchestartor.o1) -> accepting_request(engineFailure)
}
```