Formal Verification of an Automotive Scenario in Service-Oriented Computing

Maurice ter Beek¹
Stefania Gnesi¹  Nora Koch²  Franco Mazzanti¹

¹ ISTI-CNR
Pisa, Italy

² Cirquent GmbH and LMU
München, Germany

Experience Track on Automotive Systems
ICSE’08, Leipzig, Germany
Thursday 15 May 2008
Outline

1. Background
2. Automotive Scenario
3. Formal Verification
4. Lessons Learned
5. Future Work
Current State of Automotive Industry

- 80% of innovation cost of car due to software systems
- 25% of total costs of finished car due to software costs
- > 70 electronic control units (ECUs) requiring specific software

⇒ Great potential for software technologies

A. Saad (Car-IT, BMW) Javaspektrum 2/2003
Trends in Automotive Industry

Cars have an increasing number of embedded software components
- to meet emissions and fuel-economy standards
- to advance diagnostics
- to improve safety
- to reduce costs
- to add comfort and infotainment features

Requires networking of embedded software components as well as networking of car and environment

⇒ Increased system complexity and increased dependencies

“Mitsubishi Electric Corporation, IBM and ILS Technology LLC [...] are delivering a service oriented architecture (SOA) solution that is specifically designed for the automotive manufacturing industry.”

Trends in Automotive Industry

Cars have an increasing number of embedded software components
- to meet emissions and fuel-economy standards
- to advance diagnostics
- to improve safety
- to reduce costs
- to add comfort and infotainment features

Requires networking of embedded software components as well as networking of car and environment

⇒ Increased system complexity and increased dependencies

“Mitsubishi Electric Corporation, IBM and ILS Technology LLC [...] are delivering a service oriented architecture (SOA) solution that is specifically designed for the automotive manufacturing industry.”

Trends in Automotive Industry

Cars have an increasing number of embedded software components

- to meet emissions and fuel-economy standards
- to advance diagnostics
- to improve safety
- to reduce costs
- to add comfort and infotainment features

Requires networking of embedded software components as well as networking of car and environment

⇒ Increased system complexity and increased dependencies

“Mitsubishi Electric Corporation, IBM and ILS Technology LLC [...] are delivering a service oriented architecture (SOA) solution that is specifically designed for the automotive manufacturing industry.”

Formal verification of SOC applications

Service-Oriented Computing

SOC is an emerging paradigm for developing loosely coupled, interoperable, evolvable systems and applications, exploiting the pervasiveness of the Internet and its related technologies.

Goal of our research

Develop formal reasoning mechanisms and analytical tools for checking that the services resulting from a composition meet desirable correctness properties and do not manifest unexpected behaviours.

SENSORIA: http://www.sensoria-ist.eu/

Work performed in context of EU project SENSORIA: Develop a comprehensive and pragmatic (but theoretically well-founded) approach to software engineering for service-oriented systems.
Formal verification of SOC applications

Service-Oriented Computing

SOC is an emerging paradigm for developing loosely coupled, interoperable, evolvable systems and applications, exploiting the pervasiveness of the Internet and its related technologies.

Goal of our research

Develop formal reasoning mechanisms and analytical tools for checking that the services resulting from a composition meet desirable correctness properties and do not manifest unexpected behaviours.

SENSORIA: http://www.sensoria-ist.eu/

Work performed in context of EU project SENSORIA: Develop a comprehensive and pragmatic (but theoretically well-founded) approach to software engineering for service-oriented systems.
On the road, a vehicle’s diagnostic system reports a low oil level.

Triggers in-vehicle diagnostic system: car no longer driveable, send diagnostic data and car’s GPS coordinates to repair server.

Given driver’s preferences, service discovery system identifies and selects appropriate services (garage/tow truck/rental car) in area.

When driver makes appointment with garage, results of in-vehicle diagnosis are automatically sent along, allowing garage to identify the spare parts needed to repair car.

Similarly, when driver orders tow truck and rental car, the car’s GPS coordinates are sent along.

Obviously, driver is required to deposit a security payment before being able to order any service.

Each service can be denied or cancelled, causing compensation.
Extension with stereotypes and constraints specified in OCL


http://www.uml4soa.eu/

SOA structure diagrams: «service»/«service interface»/«service description»

Stereotypes for service interactions, long-running transactions, and their compensation and exception handling:

- «scope» is a structured activity that groups actions
- «compensation edge»/«exception edge» to link scopes and handlers
- «send»/«received»/«send and receive» for send and receive actions
- «compensate»/«compensate all» to trigger the execution of the compensation defined for a scope or activity; compensation is called on all subscopes in reverse order of completion
Extension with stereotypes and constraints specified in OCL

http://www.uml4soa.eu/

SOA structure diagrams: «service»/«service interface»/«service description»

Stereotypes for service interactions, long-running transactions, and their compensation and exception handling:

- «scope» is a structured activity that groups actions
- «compensation edge»/«exception edge» to link scopes and handlers
- «send»/«received»/«send and receive» for send and receive actions
- «compensate»/«compensate all» to trigger the execution of the compensation defined for a scope or activity; compensation is called on all subscopes in reverse order of completion
Components of UML Specification

UML 2.0 specification using UML4SOA

N. Koch and D. Berndl, Requirements Modelling and Analysis of Selected Scenarios of the Automotive Case Study. SENSORIA Deliverable 8.2a, Sept. 2007

- Engine: causes low oil level alert
- Discovery engine: discovers services needed
- Reasoner: selects best services
- Orchestrator: composes services to achieve goal
- Driver: calls garage, tow truck, rental car and bank
- GPS: sends vehicle’s current coordinates to services
- Garage: receives diagnostic data about vehicle
- Tow truck: receives GPS coordinates of vehicle
- Rental car: receives GPS coordinates of driver
- Bank: receives deposit from driver
Components of UML Specification

UML 2.0 specification using UML4SOA

N. Koch and D. Berndl, Requirements Modelling and Analysis of Selected Scenarios of the Automotive Case Study. SENSORIA Deliverable 8.2a, Sept. 2007

- Engine: causes low oil level alert
- Discovery engine: discovers services needed
- Reasoner: selects best services
- Orchestrator: composes services to achieve goal
- Driver: calls garage, tow truck, rental car and bank
- GPS: sends vehicle’s current coordinates to services
- Garage: receives diagnostic data about vehicle
- Tow truck: receives GPS coordinates of vehicle
- Rental car: receives GPS coordinates of driver
- Bank: receives deposit from driver
Operational Model: Communicating State Machines

Subcomponent BankCommunication

- BankComm (Defers bankrevoke)
- b1
  - bankcharge / bank.requestCardCharge(...)
  - chargeResponseFail / self.bankFail
- b2
  - chargeResponseOK(chargeID) / chargedID := chargeID; self.bankOK
- b3
  - bankrevoke / bank.revokeCardCharge(self,chargedID)
- b4
- b5
- b6
  - bankrevoke
  - bankrevokeOK
Modelling Assumptions

- We do not define a separate state machine for each component, but rather structure some of them as subcomponents of others.
- We abstract altogether from a remote discovery state machine (to search for services in a remote repository).
- `LocalDiscovery` returns at most one choice of services.
- Compensations are explicitly modelled as requests to cancel operations (viz. `bankrevoke` and `garagerevoke`).
- All communications between `Car`'s subcomponents, and those between these subcomponents and `Bank` and `RoadAssistance` (i.e. all service invocations), modelled as request/response pairs (whereas this is necessary for the former (synchronous operation calls would deadlock), the latter might equally well be modelled as synchronous operation calls).
Validation by hand is not feasible: 535 states and 814 transitions
Model Checking

Model checking

Automatic analysis of correctness properties of system designs; such verifications are exhaustive, i.e. all possible input combinations and states are taken into account, and a counterexample is usually generated in case a certain property does not hold

Correctness properties: (un)desired system behaviour

- Qualitative: functional properties (considered in this paper)
- Quantitative: performance/dependability properties (future)

Advantage: detect design errors before implementation

Design errors — which constitute up to 40% of software errors and are among the most expensive ones to resolve if discovered after implementation [LRRA98] — can be detected in the design phase, leading to considerable reductions in cost and to improved quality.
Model Checking

Model checking

Automatic analysis of correctness properties of system designs; such verifications are exhaustive, i.e. all possible input combinations and states are taken into account, and a counterexample is usually generated in case a certain property does not hold.

Correctness properties: (un)desired system behaviour

- Qualitative: functional properties (considered in this paper)
- Quantitative: performance/dependability properties (future)

Advantage: detect design errors before implementation

Design errors — which constitute up to 40% of software errors and are among the most expensive ones to resolve if discovered after implementation [LRRA98] — can be detected in the design phase, leading to considerable reductions in cost and to improved quality.
Model Checking

Model checking
Automatic analysis of correctness properties of system designs; such verifications are exhaustive, i.e. all possible input combinations and states are taken into account, and a counterexample is usually generated in case a certain property does not hold.

Correctness properties: (un)desired system behaviour
- Qualitative: functional properties (considered in this paper)
- Quantitative: performance/dependability properties (future)

Advantage: detect design errors before implementation
Design errors — which constitute up to 40% of software errors and are among the most expensive ones to resolve if discovered after implementation [LRRA98] — can be detected in the design phase, leading to considerable reductions in cost and to improved quality.
The Need for Action- and State-based Logics

Model checking primarily evolved over Kripke structures

- Information associated to states (e.g. SMV, NuSMV, etc.)
- This approach is particularly amenable to modelling hardware

Theory of concurrency evolved over Labelled Transition Systems

- Information associated to transitions (CCS, LOTOS, $\pi$-calculus, etc.)
- Many aspects of software (in particular the most interesting ones for reactive, concurrent and distributed software) are events, i.e. actions

The emerging trend in industry is to use UML (state diagrams)

- Ease of expressiveness w.r.t. pure action- or state-based logics
- Their use often leads to a reduced state space, small memory needs and less time spent for verification
The Need for Action- and State-based Logics

Model checking primarily evolved over Kripke structures
Information associated to states (e.g. SMV, NuSMV, etc.)
This approach is particularly amenable to modelling hardware

Theory of concurrency evolved over Labelled Transition Systems
Information associated to transitions (CCS, LOTOS, π-calculus, etc.)
Many aspects of software (in particular the most interesting ones for reactive, concurrent and distributed software) are events, i.e. actions

The emerging trend in industry is to use UML (state diagrams)

- Ease of expressiveness w.r.t. pure action- or state-based logics
- Their use often leads to a reduced state space, small memory needs and less time spent for verification
### The Need for Action- and State-based Logics

<table>
<thead>
<tr>
<th>Model checking primarily evolved over Kripke structures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Information associated to states (e.g. SMV, NuSMV, etc.)</td>
</tr>
<tr>
<td>This approach is particularly amenable to modelling hardware</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Theory of concurrency evolved over Labelled Transition Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>Information associated to transitions (CCS, LOTOS, π-calculus, etc.)</td>
</tr>
<tr>
<td>Many aspects of software (in particular the most interesting ones for reactive, concurrent and distributed software) are events, i.e. actions</td>
</tr>
</tbody>
</table>

To use the full potential of specification languages (like UML) that allow both action and state changes to be modelled, various action- and state-based logics have been developed:

- Ease of expressiveness w.r.t. pure action- or state-based logics
- Their use often leads to a reduced state space, small memory needs and less time spent for verification
The Need for Action- and State-based Logics

Model checking primarily evolved over Kripke structures
Information associated to states (e.g. SMV, NuSMV, etc.)
This approach is particularly amenable to modelling hardware

Theory of concurrency evolved over Labelled Transition Systems
Information associated to transitions (CCS, LOTOS, π-calculus, etc.)
Many aspects of software (in particular the most interesting ones for reactive, concurrent and distributed software) are events, i.e. actions

To use the full potential of specification languages (like UML) that allow both action and state changes to be modelled, various action- and state-based logics have been developed

- Ease of expressiveness w.r.t. pure action- or state-based logics
- Their use often leads to a reduced state space, small memory needs and less time spent for verification
Temporal Logic UCTL

UCTL: Action- and State-based Branching-time Temporal Logic


- UCTL defined as extension of action-based CTL (ACTL [DNV90])
- UCTL includes both CTL and ACTL
- All possible system evolutions are formally represented as a *Doubly-labelled Transition System (L²TS)* [DNV95]
  - states represent the various system configurations
  - edges represent the possible evolutions of a system configuration

The service-oriented logic SOCL is a recent specialization of UCTL meant to capture peculiar aspects of services [FGLMPT08]
Temporal Logic UCTL

UCTL: Action- and State-based Branching-time Temporal Logic


- UCTL defined as extension of action-based CTL (ACTL [DNV90])
- UCTL includes both CTL and ACTL
- All possible system evolutions are formally represented as a *Doubly-labelled Transition System* (L²TS) [DNV95]
  - states represent the various system configurations
  - edges represent the possible evolutions of a system configuration

The service-oriented logic SOCL is a recent specialization of UCTL meant to capture peculiar aspects of services [FGLMPT08]
Temporal Logic UCTL

UCTL: Action- and State-based Branching-time Temporal Logic


- UCTL defined as extension of action-based CTL (ACTL [DNV90])
- UCTL includes both CTL and ACTL
- All possible system evolutions are formally represented as a Doubly-labelled Transition System (L²TS) [DNV95]
  - states represent the various system configurations
  - edges represent the possible evolutions of a system configuration

The service-oriented logic SOCL is a recent specialization of UCTL meant to capture peculiar aspects of services [FGLMPT08]
### UCTL: Syntax

#### Action formulae syntax
Let $Act$ be a set of observable events. Then the language of event formulae on $Act \cup \{\tau\}$ is defined as follows:

\[
\chi ::= \text{true} \mid e \mid \tau \mid \neg \chi \mid \chi \land \chi
\]

#### UCTL syntax

**(state formulae)** $\phi ::= \text{true} \mid \pi \mid \neg \phi \mid \phi \land \phi' \mid E\psi \mid A\psi$

**(path formulae)** $\psi ::= X_\gamma \phi \mid \phi \chi U \phi' \mid \phi \chi U_\gamma \phi' \mid \phi \chi W \phi' \mid \phi \chi W_\gamma \phi'$

#### Some derived modalities

$< \gamma > \phi$ stands for $EX_\gamma \phi$

$EF\phi$ stands for $E(\text{true}_built \land U_\phi)$

$[\gamma] \phi$ stands for $\neg < \gamma > \neg \phi$

$AG\phi$ stands for $\neg EF \neg \phi$
UCTL: Syntax

Action formulae syntax

Let $Act$ be a set of observable events. Then the language of event formulae on $Act \cup \{\tau\}$ is defined as follows:

$$
\chi \ ::= \ true \mid e \mid \tau \mid \neg \chi \mid \chi \land \chi
$$

UCTL syntax

(state formulae) $\phi \ ::= \ true \mid \pi \mid \neg \phi \mid \phi \land \phi' \mid E\psi \mid A\psi$

(path formulae) $\psi \ ::= \ X_\gamma \phi \mid \phi \land U \phi' \mid \phi \land U_\gamma \phi' \mid \phi \land W \phi' \mid \phi \land W_\gamma \phi'$

Some derived modalities

$< \gamma > \phi$ stands for $\text{EX}_\gamma \phi$

$EF \phi$ stands for $E(\text{true}_\text{true} \land U \phi)$

$[\gamma] \phi$ stands for $\neg < \gamma > \neg \phi$

$AG \phi$ stands for $\neg EF \neg \phi$
Action formulae syntax

Let $Act$ be a set of observable events. Then the language of event formulae on $Act \cup \{\tau\}$ is defined as follows:

$$\chi ::= true \mid e \mid \tau \mid \neg \chi \mid \chi \land \chi$$

UCTL syntax

\begin{align*}
\text{(state formulae)} & \quad \phi ::= true \mid \pi \mid \neg \phi \mid \phi \land \phi' \mid E\psi \mid A\psi \\
\text{(path formulae)} & \quad \psi ::= X_{\gamma}\phi \mid \phi \chi U \phi' \mid \phi \chi U_{\gamma} \phi' \mid \phi \chi W \phi' \mid \phi \chi W_{\gamma} \phi'
\end{align*}

Some derived modalities

\begin{align*}
< \gamma > \phi & \text{ stands for } EX_{\gamma} \phi \\
EF \phi & \text{ stands for } E(true \land U\phi) \\
[\gamma] \phi & \text{ stands for } \neg < \gamma > \neg \phi \\
AG \phi & \text{ stands for } \neg EF \neg \phi
\end{align*}
We have developed on-the-fly model checkers to verify UCTL/SocL formulae (specifying both action- and state-based properties) over a set of communicating UML state machines.

Current UMC/CMC prototypes can be experimented via a web interface:

http://fmt.isti.cnr.it/{u/c}mc

We have used UMC v3.4 on an ordinary PC to verify a number of behavioural properties expressed in UCTL over our implementation of the on road assistance scenario.

The verifications show that the requirements model of the on road assistance scenario has been well designed in [KB07].
We have developed on-the-fly model checkers to verify UCTL/SocL formulae (specifying both action- and state-based properties) over a set of communicating UML state machines.

Current UMC/CMC prototypes can be experimented via a web interface:

http://fmt.isti.cnr.it/{u/c}mc

We have used UMC v3.4 on an ordinary PC to verify a number of behavioural properties expressed in UCTL over our implementation of the on road assistance scenario.

The verifications show that the requirements model of the on road assistance scenario has been well designed in [KB07].
A service is *responsive* if it guarantees a response to each received request.

If *Car* requests *Bank* to charge a credit card, then *Bank* will reply by notifying either a successful or a failed attempt to charge the credit card.

**UCTL property F1**

\[ AG \left[ \text{requestCardCharge} \right] \\
A \left[ \text{true} \right. \left. U \text{chargeResponseOK} \lor \text{chargeResponseFail} \right] \]
A service is *responsive* if it guarantees a response to each received request.

If **Car** requests **Bank** to charge a credit card, then **Bank** will reply by notifying either a successful or a failed attempt to charge the credit card.

UCTL property F1:

\[ \text{AG} \left[ \text{requestCardCharge} \right] \rightarrow \text{A} \left[ \text{true} \rightarrow \text{U} \left[ \text{chargeResponseOK} \lor \text{chargeResponseFail} \rightarrow \text{true} \right] \right] \]
A service is *responsive* if it guarantees a response to each received request.

If **Car** requests **Bank** to charge a credit card, then **Bank** will reply by notifying either a successful or a failed attempt to charge the credit card.

**UCTL property F1**

\[ \mathit{AG\ [\text{requestCardCharge}]} \]

\[ \mathit{A\ [\text{true true U chargeResponseOK} \lor \text{chargeResponseFail} \text{ true}]} \]
A service is *reliable* if it guarantees a successful response whenever it accepts a request (for this service).

Reservation requests from *Car* to *Garage* are always followed by a notification of success.

**UCTL property F3**

\[
AG \ [ \text{requestGarage} ] \\
A \ [ \text{true} \ \text{true} \ U_{\text{garageResponseOK}} \ \text{true} ]
\]
A service is *reliable* if it guarantees a successful response whenever it accepts a request (for this service)

Reservation requests from **Car** to **Garage** are always followed by a notification of success

UCTL property F3

\[ AG \ [ \text{requestGarage} ] \]
\[ A \ [ \text{true} \text{ true} U \text{garageResponseOK} \text{ true} ] \]
A service is *reliable* if it guarantees a successful response whenever it accepts a request (for this service).

Reservation requests from **Car** to **Garage** are always followed by a notification of success.

UCTL property F3

\[
AG \ [ \text{requestGarage} ] \\
A \ [ \text{true} \ \text{true} \ U_{\text{garageResponseOK}} \ \text{true} ]
\]
A service guarantees *uniqueness of response* if it guarantees a single successful response whenever it accepts a request (for this service).

After a reservation request from *Car* to *Garage*, it cannot happen that the *Car* receives more than one notification.

**UCTL property F4**

\[ \text{AG } [\text{requestGarage}] \text{ AF } \langle \text{garageResponse} \rangle \quad \neg \text{EF } \langle \text{garageResponse} \rangle \text{ true} \]
Uniqueness of Response

A service guarantees *uniqueness of response* if it guarantees a single successful response whenever it accepts a request (for this service)

After a reservation request from **Car** to **Garage**, it cannot happen that the **Car** receives more than one notification

UCTL property F4

\[ AG[\text{requestGarage}] \ AF \ <\text{garageResponse}> \ \neg EF \ <\text{garageResponse}> \ true \]
A service guarantees *uniqueness of response* if it guarantees a single successful response whenever it accepts a request (for this service).

After a reservation request from *Car* to *Garage*, it cannot happen that the *Car* receives more than one notification.

**UCTL property F4**

\[ AG \text{[} requestGarage \text{]} \implies AF \text{[} garageResponse \text{]} \land \neg EF \text{[} garageResponse \text{]} \text{true} \]
Table: Validation results.

<table>
<thead>
<tr>
<th>formula</th>
<th>property</th>
<th>validity</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>service responsiveness</td>
<td>true</td>
</tr>
<tr>
<td>F2</td>
<td>service coordination</td>
<td>true</td>
</tr>
<tr>
<td>F3</td>
<td>service reliability</td>
<td>false</td>
</tr>
<tr>
<td>F4</td>
<td>uniqueness of response</td>
<td>true</td>
</tr>
<tr>
<td>F5</td>
<td>functional system behaviour</td>
<td>true</td>
</tr>
</tbody>
</table>

⇒ Not surprising that F3 is false: Garage might be temporarily unable to provide the requested service and thus send the unsuccessful response `garageResponseFail`

(note that Garage is responsive: F1-like formula does hold)
Lessons Learned

- On road assistance scenario was outcome of discussions with automotive experts on possible new services for drivers to be provided by the in-vehicle computers.
- Validation shows requirements model has been well designed.
- Case study moreover shows usefulness and feasibility of a formal approach to specify and rigorously analyze a system design, also in industrial contexts.
- Room for improvement of our model checker: regarding UML support of the tool, regarding further optimizations of on-the-fly model-checking algorithm and regarding overall usability and user-interface issues.
Future Work

- Relax modelling assumptions made w.r.t. requirements models
- Verify more complex scenarios (drivers competing for tow trucks)
- Quantitative analysis of on road assistance scenario
- Improve the UCTL/SocL model checker
Future Work

- Relax modelling assumptions made w.r.t. requirements models
- Verify more complex scenarios (drivers competing for tow trucks)
- Quantitative analysis of on road assistance scenario
- Improve the UCTL/SocL model checker
Future Work

- Relax modelling assumptions made w.r.t. requirements models
- Verify more complex scenarios (drivers competing for tow trucks)
- Quantitative analysis of on road assistance scenario
- Improve the UCTL/SocL model checker
A service is *coordinated* if its confirmation is always preceded by a request (for this service)

**Car** cannot receive a notification of the fact that the credit card has been charged, if it did not previously request **Bank** to do so

UCTL property F2

$\neg E \left[ true \neg \text{requestCardCharge} U \text{chargeResponseOK} \text{true} \right]$
A service is *coordinated* if its confirmation is always preceded by a request (for this service).

**Car** cannot receive a notification of the fact that the credit card has been charged, if it did not previously request **Bank** to do so.

UCTL property F2

\[ \neg E \left[ \text{true} \quad \neg \text{requestCardCharge} \quad U \text{chargeResponseOK} \quad \text{true} \right] \]
A service is \textit{coordinated} if its confirmation is always preceded by a request (for this service)

\textbf{Car} cannot receive a notification of the fact that the credit card has been charged, if it did not previously request \textbf{Bank} to do so

\begin{align*}
\neg E[ true \neg \text{requestCardCharge} U \text{chargeResponseOK} true ]
\end{align*}
For instance: success of a service request is always followed by either its cancellation or by the success of another service request.

If **Bank** notifies **Car** of successfully charging the credit card, then in the future either this operation will be revoked or **Car** will receive a notification of the fact that **Garage** has been reserved.

UCTL property F5

\[ AG [ \text{chargeResponseOK} ] \]

\[ A [ true \ U \text{garageResponseOK} \lor \text{bankrevokeOK} \ U \text{true} ] \]
Functional System Behaviour

For instance: success of a service request is always followed by either its cancellation or by the success of another service request

If **Bank** notifies **Car** of successfully charging the credit card, then in the future either this operation will be revoked or **Car** will receive a notification of the fact that **Garage** has been reserved

UCTL property F5

\[
AG [ \text{chargeResponseOK} ] \\
A [ true \ U \text{garageResponseOK} \lor \text{bankrevokeOK} \ U \text{true} ]
\]
For instance: success of a service request is always followed by either its cancellation or by the success of another service request.

If **Bank** notifies **Car** of successfully charging the credit card, then in the future either this operation will be revoked or **Car** will receive a notification of the fact that **Garage** has been reserved.

**UCTL property F5**

\[
AG \ [ \ \text{chargeResponseOK} \ ] \\
A \ [ \ \text{true} \ \text{true} \ U \ \text{garageResponseOK} \ \lor \ \text{bankrevokeOK} \ \text{true} \ ]
\]