Semantic-Based Development of Service-Oriented Systems

Martin Wirsing

LMU München

in Kooperation mit

A. Clark[1], S. Gilmore[1], M. Hölzl[2], A. Knapp[2], N. Koch[2], A. Schroeder[2]

[1]University of Edinburgh [2]LMU München

YRSOC, Leicester, June 12, 2007

Contents

- **Motivation: Web Services**
- **SENSORIA: Systematic Development of Service-Oriented Systems**
- **Semantic-based service-oriented extension of UML**
  - Example: Orchestration with compensation
  - Semantics by model transformation to Saga process calculus
- **From requirements to design of service architectures**
  - Soft Constraints and preferences for selecting the best service
  - Orchestration design by model transformation to state diagrams
  - Model checking the orchestration design
- **Analysis of quantitative properties: Service Level Agreements**
  - Performance and scalability modelling in UML
  - Translation to PEPA
  - Analysis of Service Level Agreement
- **Concluding remarks**

Service-Oriented Sytems

Computing is becoming a utility and software a service. [. . .] applications will no longer be a big chunk of software that runs on a computer but a combination of web services; and the platform for which developers write their programs will no longer be the operating system, but application servers.

[The Economist, May 2003]

- Selling services has become the biggest growth business in the IT industry
  - changes the economics of IT industry and
  - influences the e-Society as a whole.
- Today, services are being delivered through the

  Web, Personal Digital Assistants, mobile phones…
- Tomorrow, they will be delivered on all kinds of

  global computers.

Service-oriented Systems

- **Service**

  autonomous, **platform-independent** computational entity

  that can be

  described, published, categorised, discovered.

- **Services** can be **dynamically assembled** for developing

  massively **distributed, interoperable, evolvable** systems and applications.

- **Service-Oriented Computing**
  - addressed by IT industry only in an ad-hoc and undisciplined way
  - theoretical foundations are missing, but needed for
    - trusted interoperability,
    - predictable compositionality,
    - ensuring adequate software quality.
- How can one guarantee

  correctness, security and appropriate resources usage of services

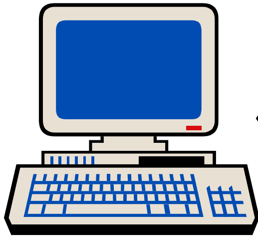
  if service discovery and negotiation occur without human intervention?

The *SENSORIA* Project

- **IST-FET Integrated Project** 2005-2009
  - Coordinator: LMU München
  - 18 Partners: U. Pisa, Florence, Bologna, Trento, Leicester, Edinburgh, Imperial College, University College, Lisbon, Warsaw, Budapest, DTU, ISTI Pisa, Poli Milano, Telecom Italia, FAST, S&N, ATX
- Novel comprehensive approach to
  Engineering of software systems for
  Service-Oriented Overlay Computers
  integrating
  - foundational theories, techniques, and methods and
  - pragmatic software engineering
- **Application areas**
  - e-business
  - **automotive systems**
  - e-learning
  - telecommunications

A Typical Scenario for SENSORIA Service Design

Designers Interface



UML for GC Services
(Model-driven Development)

Automatic connections

Hidden from the designer

Mathematical Models
(Primitives and languages for GC -services)

Integration
Simulation/verification

Quantitative and Qualitative Properties
(Performance, reliability, fault-tolerance, security, trust, mobility, …)

SENSORIA Development

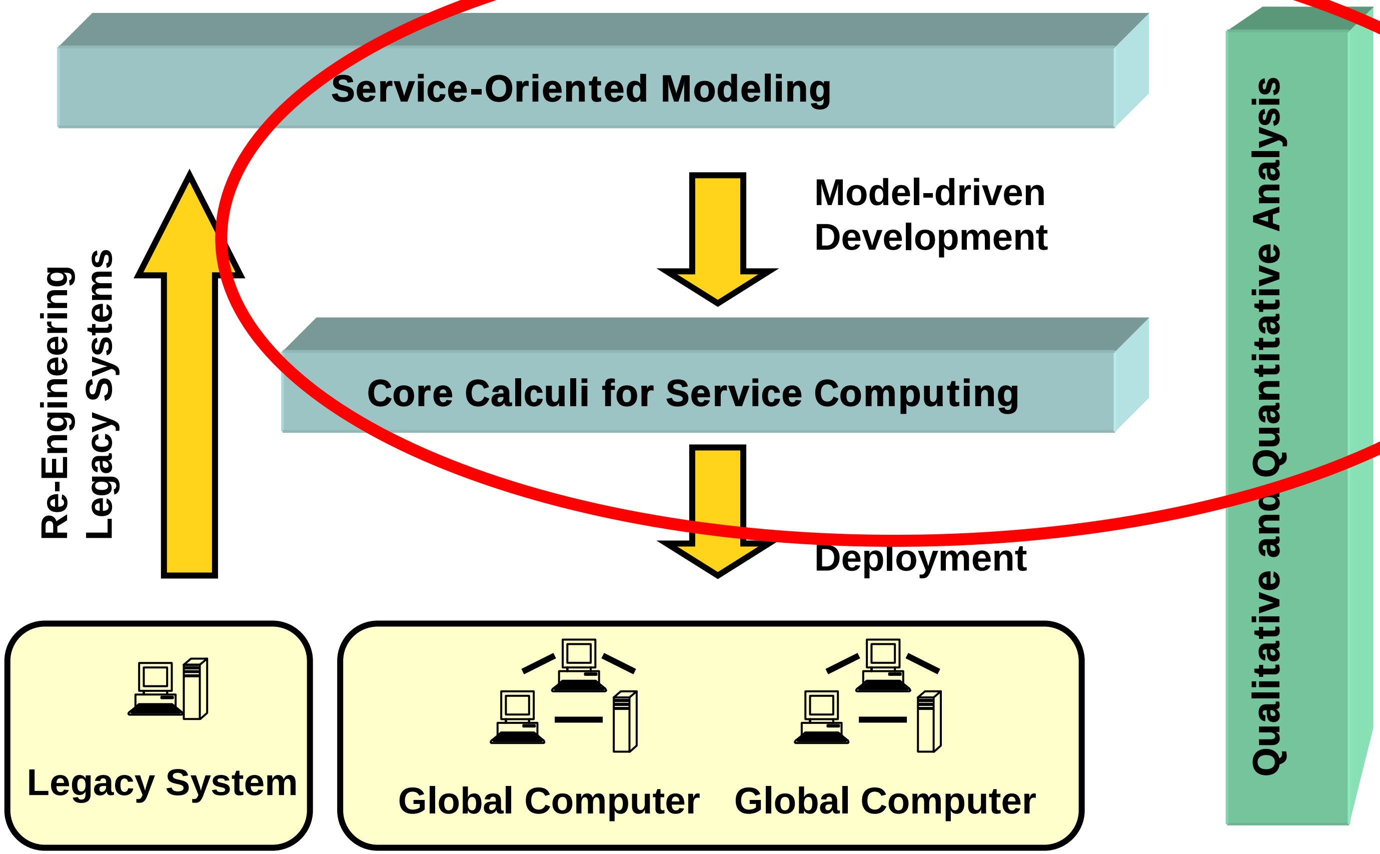
integrates

practical SW Engineering

with

math. foundations

SENSORIA Detailed Approach

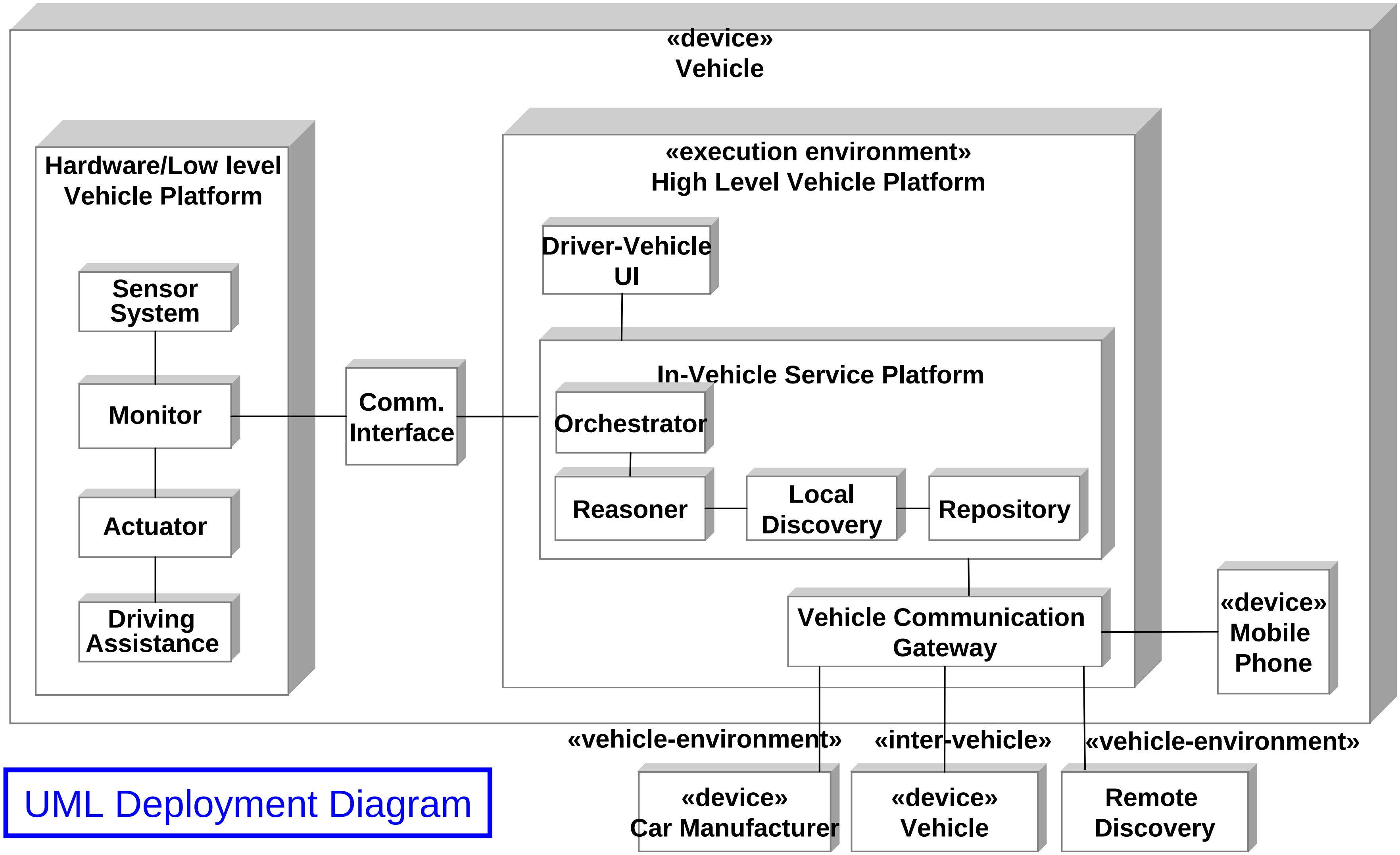


Automotive Case Study

- **More and more embedded computers in cars**

  - **Safety-critical software** (ESP, …)
  - **Infotainment** (e.g. office in the car)
  - **E-assistance for accidents and car breakdown**
    - **Discovering and booking tow truck service, garage, and rental car in the area**
    - **Sending an ambulance in case the driver does not answer after an accident**

Simplified SW-Architecture for the Car



UML Deployment Diagram

SENSORIA

Example: Car Repair Scenario

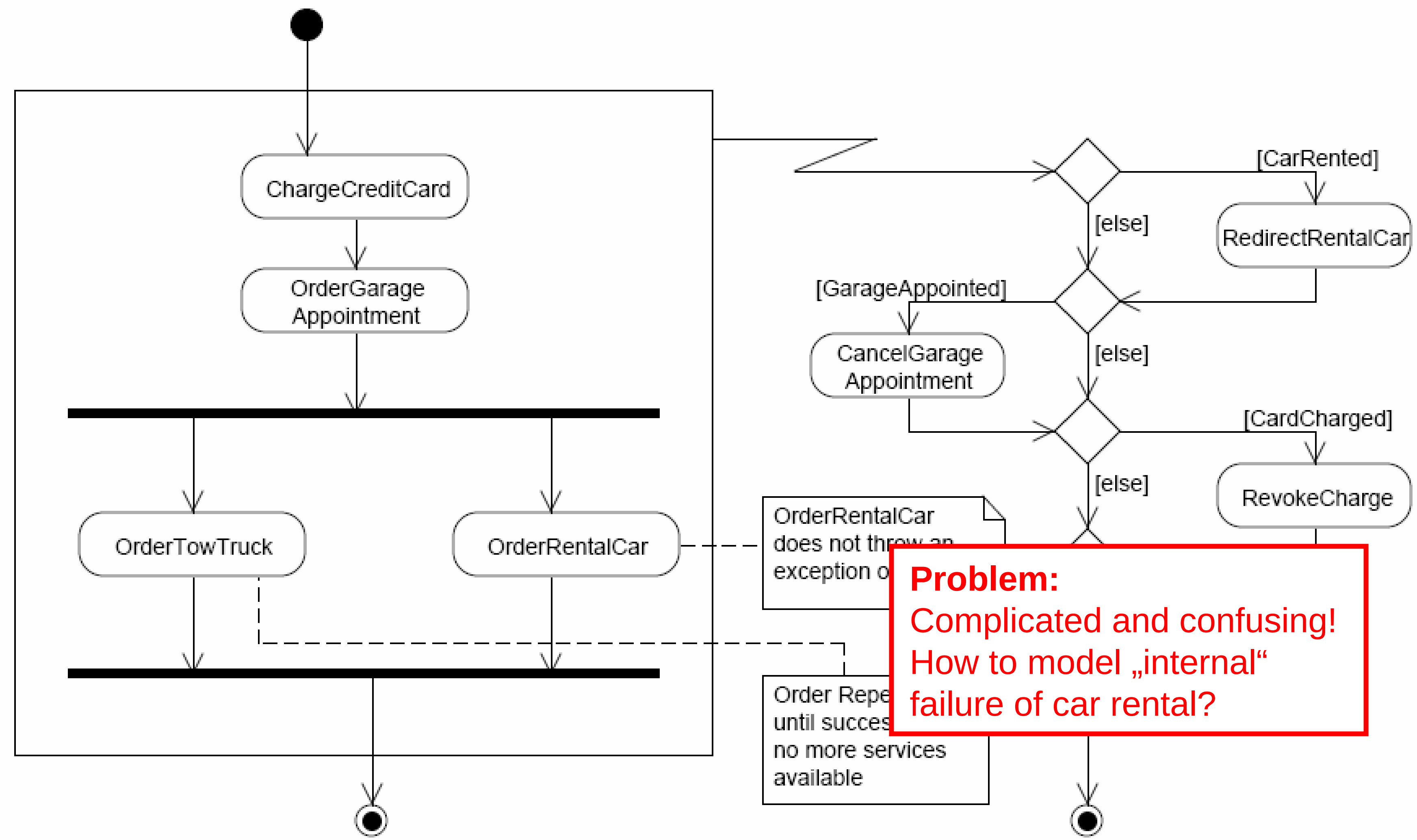
Ludwig-Maximilians-Universität München

LMU

- **The diagnostic system reports a severe failure in the car engine so that the car is no longer drivable.**
- **The car's discovery system identifies garages, car rentals and towing truck services in the car's vicinity.**
- **The in-vehicle service platform selects a set of adequate offers taking into account personalised policies and preferences of the driver and tries to order them. The owner of the car has to deposit a security payment before being able to order any services.**
- **In case of failure compensation is needed:**
  - If ordering a garage fails, the tow truck has to be cancelled as well and the rental car has to be sent to the breakdown location.
  - If ordering a tow truck fails, the garage appointment has to be cancelled as well.
  - Failure of renting a car does not influence
  - the booking of garage and tow truck.

➔ **„Long running transactions"of services require compensation techniques**

Modelling Compensation in „Classic“ UML

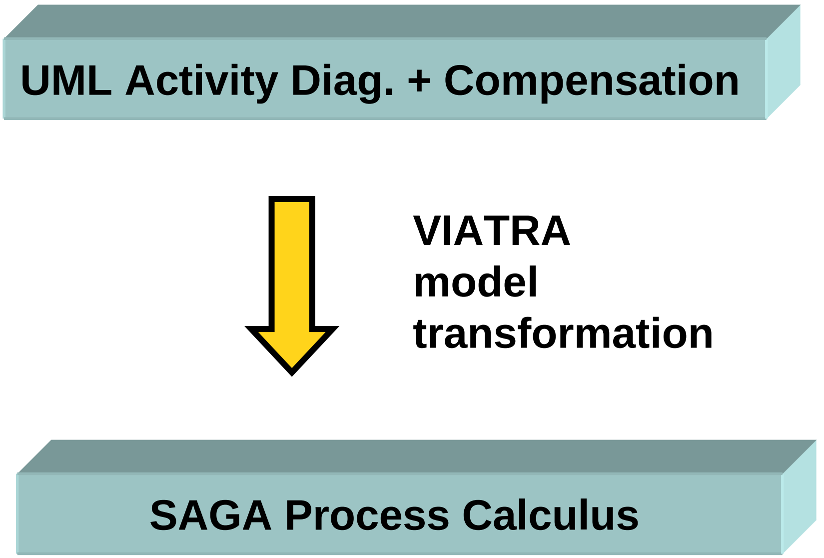


Problem:
Complicated and confusing!
How to model „internal“ failure of car rental?

SENSORIA Approach to Long Running Transactions

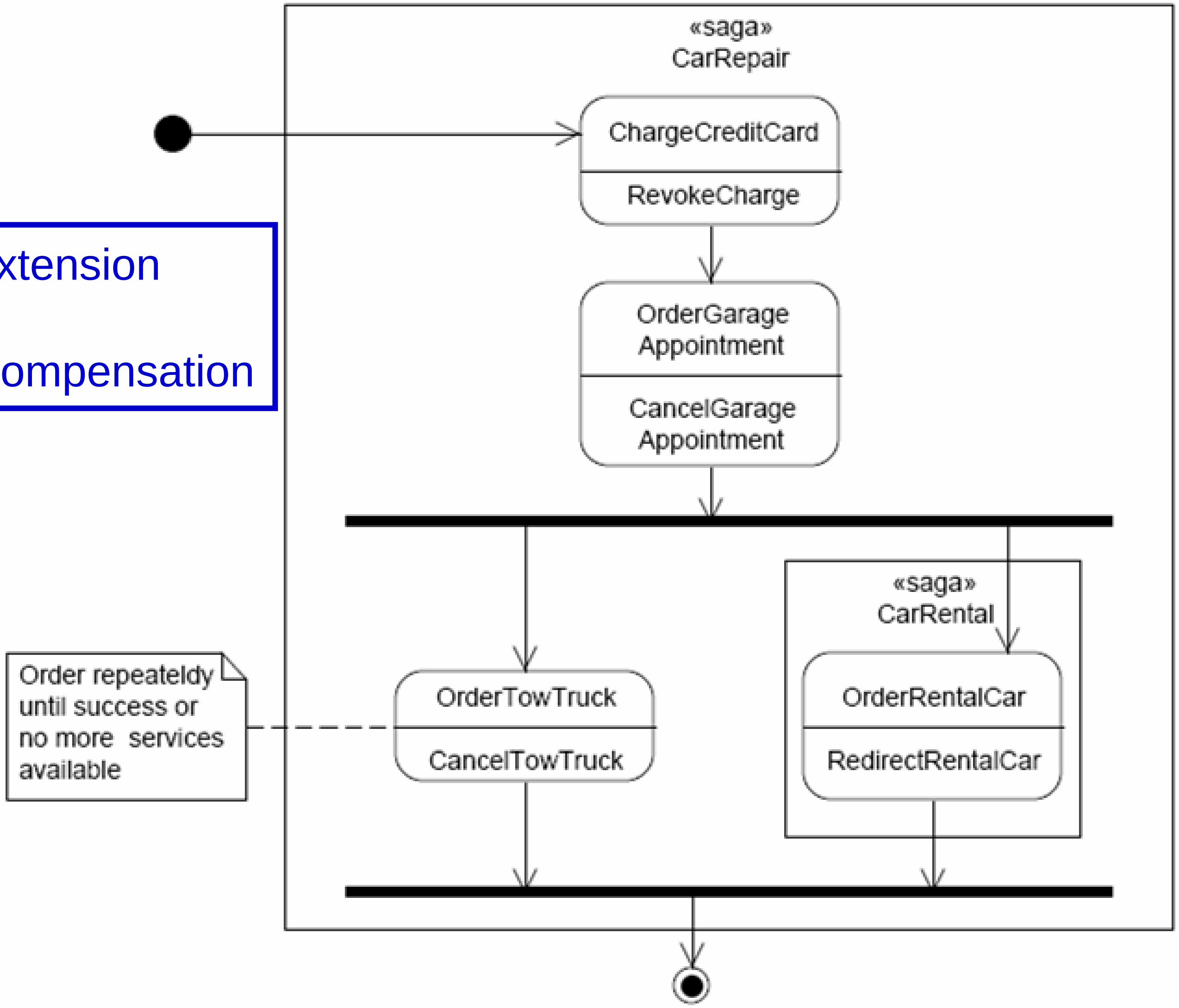
SENSORIA Approach:

- **Extend UML by notations for long running transactions**
- **Use formal models to derive semantics of UML extensions:**
  - The Saga process calculus supports the formal treatment of compensation [Bruni, Montanari et. al. 2005]
  - **Extend UML by Sagas**
  - **Define semantics by model transformations**



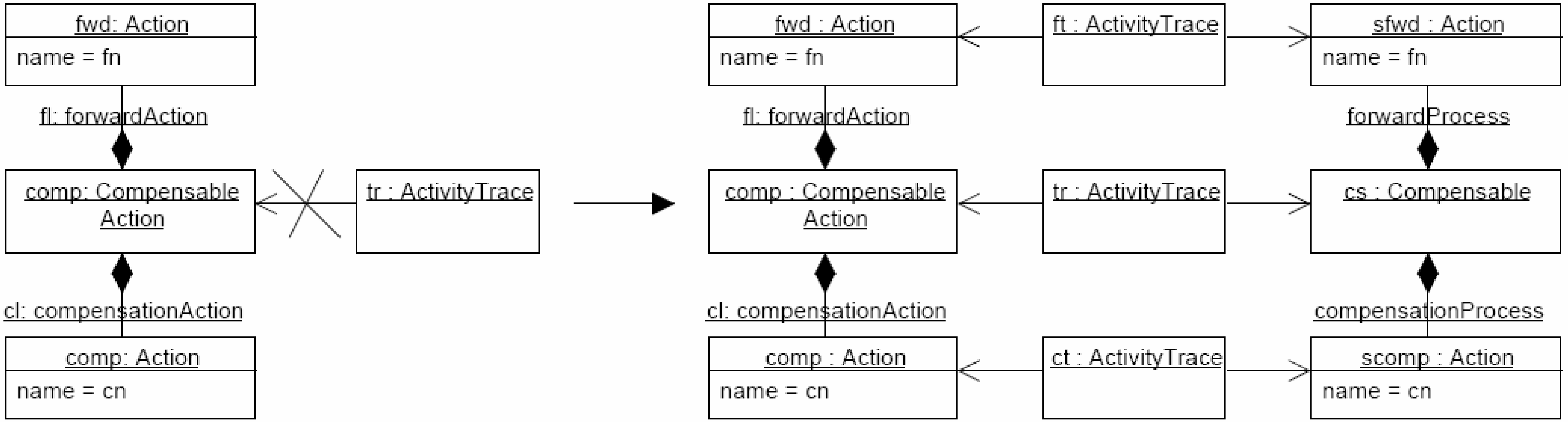
Saga Compensation in UML

UML-extension
by
Saga compensation



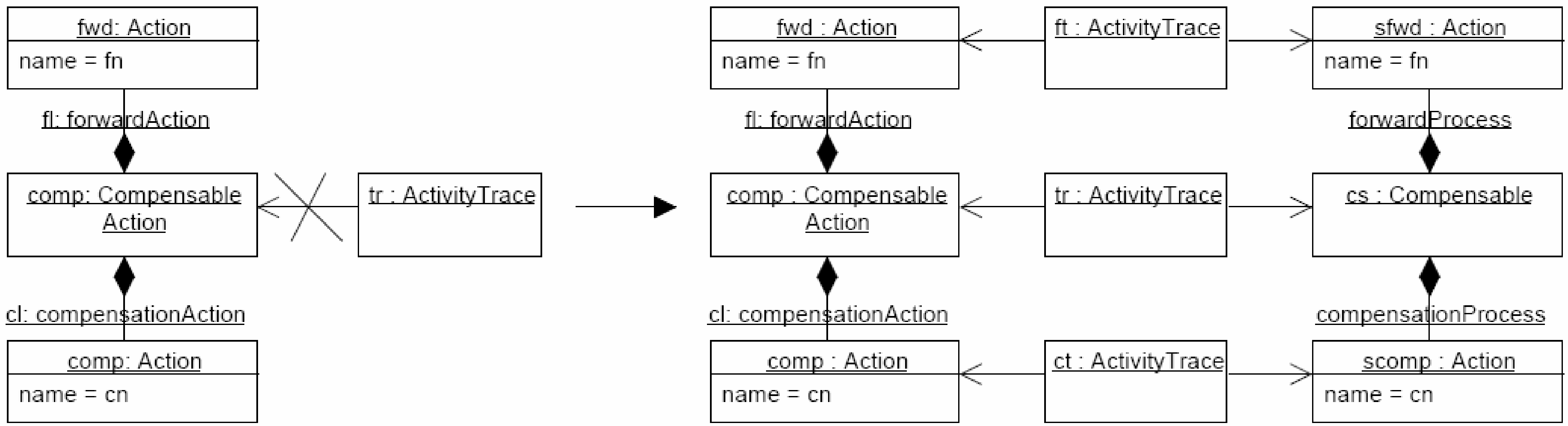
(Meta-) Model Transformation: UML into Sagas

VIATRA2 [Varro et al.] Graph transformation for compensable actions:



(Meta-) Model Transformation: UML into Sagas

VIATRA2 [Varro et al.] graph transformation for compensable actions:



- **Transforming the UML activity diagram yields SAGA program:**

```
(ChargeCCard % RevokeCharge) ;
(OrderGar % CancelGar) ;
((OrderTTruck % CancelTTruck) | [OrderCar % RedirectCar])
```

- **Semantics of UML extension is defined by SAGA semantics**

From Requirements to Design of Service Architectures

- **Requirements**
  - Define (workflow) scenarios and model them by UML (e.g. Activity Diagrams)
  - Identify and specify services
  - Specify required qualitative and quantitative properties
    (Constraints, preferences, global service level agreements, …)
- **Design**
  - Specify service architecture
  - Derive service selection, orchestration and design of services from requirements by model transformation
  - Analyse design by mathematical techniques (model checking, Markov chains, ..)
- **Examples**
  - **Car Repair Scenario**: Soft constraints and preferences,
    orchestration design and model checking of the design
  - **Road Accident Scenario**: UML State Diagram with performance annotation,
    SLA and validation of the SLA

Car Repair Scenario: Soft Constraints and Preferences for Services

- **Identify services:**
  - Order garage, tow truck, and rental car
- **Choosing the „best“ offer**
  - **Approach**: **Soft Constraints over C-Semirings** [Bistarelli, Montanari, Rossi 97]
    Policy language with preferences [W, Hölzl 06a, b]
  - **Example constraints and preferences**
    - **Repair as soon as possible, in less than 48 hours**

$$\mathit{fastRepair} : [\mathit{garage\text{-}duration} \mid n \mapsto \lfloor 48/n \rfloor]$$

Fuzzy ring: 0 is the minimum

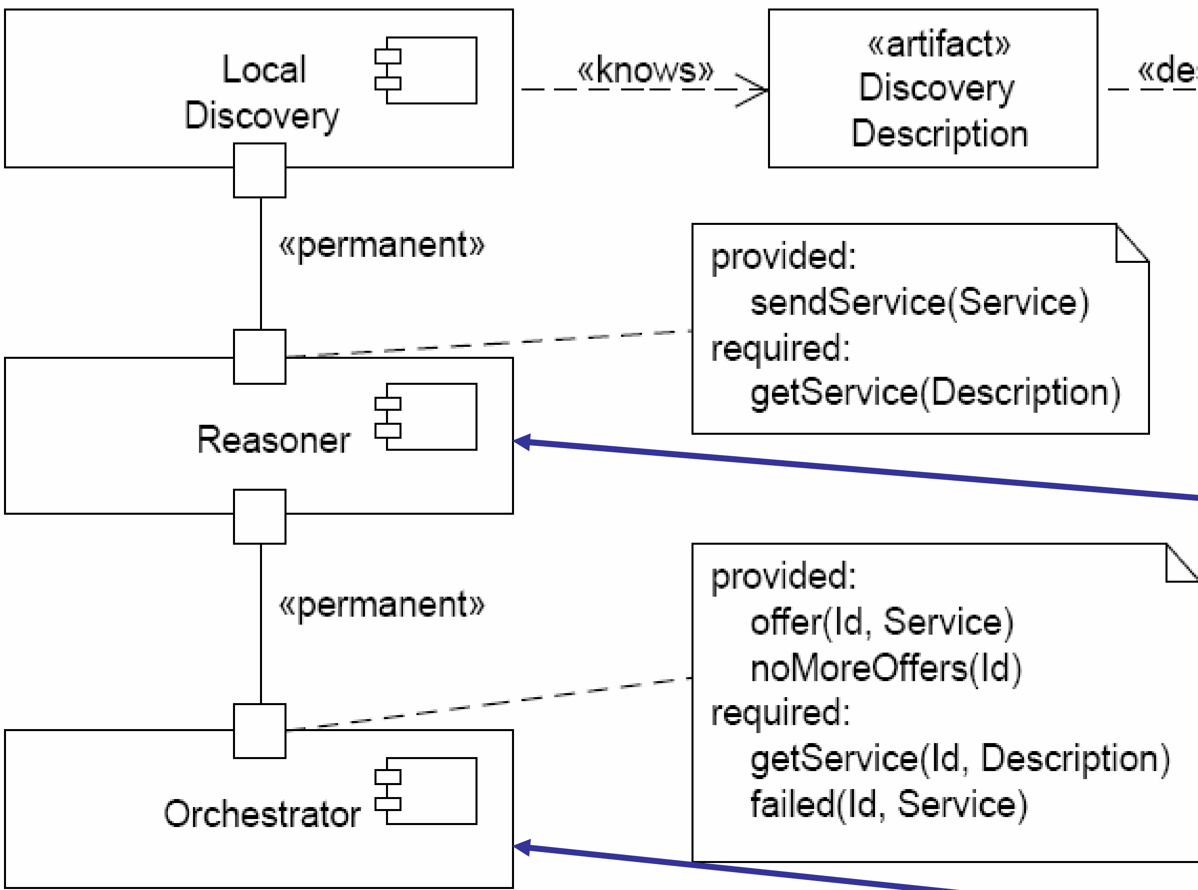
    - **Private repair as cheap as possible, 1000 Euro and more almost unacceptable**

$$\mathit{cheapRepair} : \text{in context } \neg \mathit{work\text{-}related}?$$
$$\text{assert } [\mathit{garage\text{-}cost} \mid n \mapsto \lceil 1000/n \rceil] \text{ end}$$

    - **Preference: Prefer fast repair to cheap repair**

$$\mathit{fastRepair} > \mathit{cheapRepair}$$

Design:
Components of High-Level Vehicle Platform



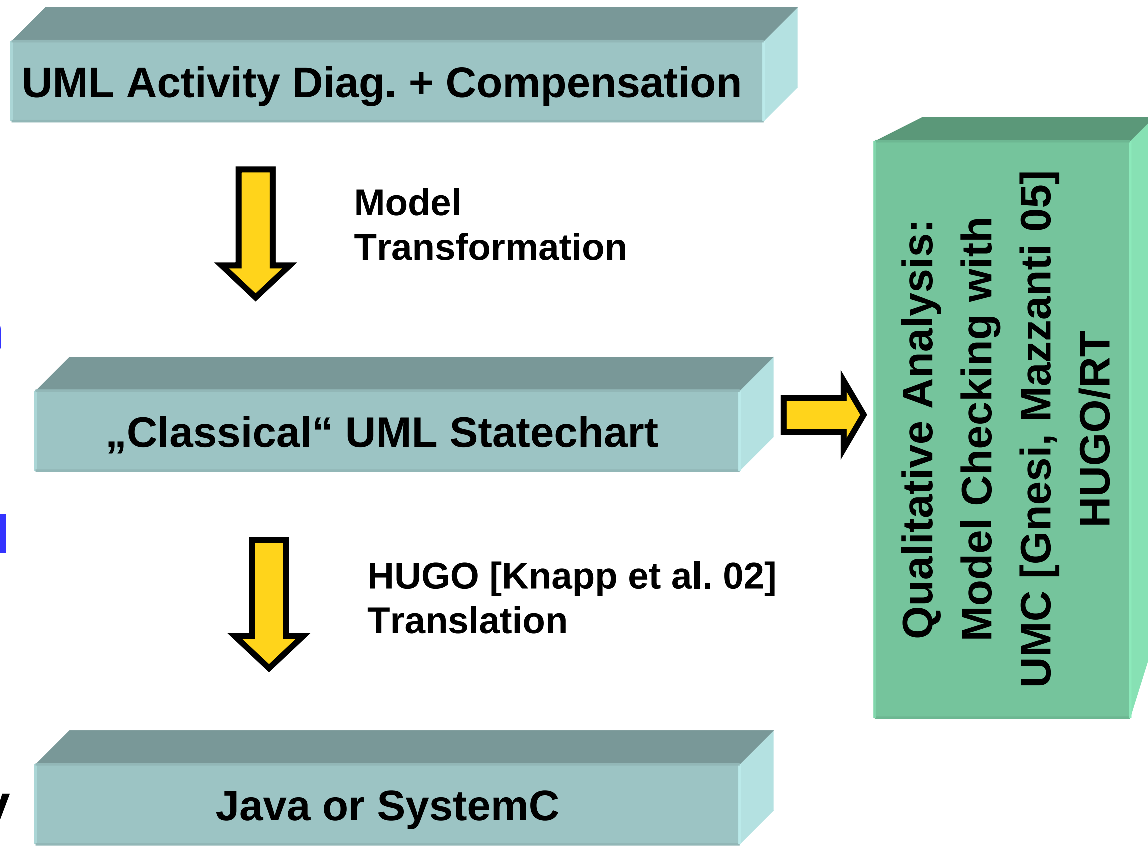
Reasoner: computes best/acceptable solutions of constraints and preferences with Soft Constraint Solver
[Frühwirth 02, W et al. 06, ..

Orchestrator: realizes workflow requirements

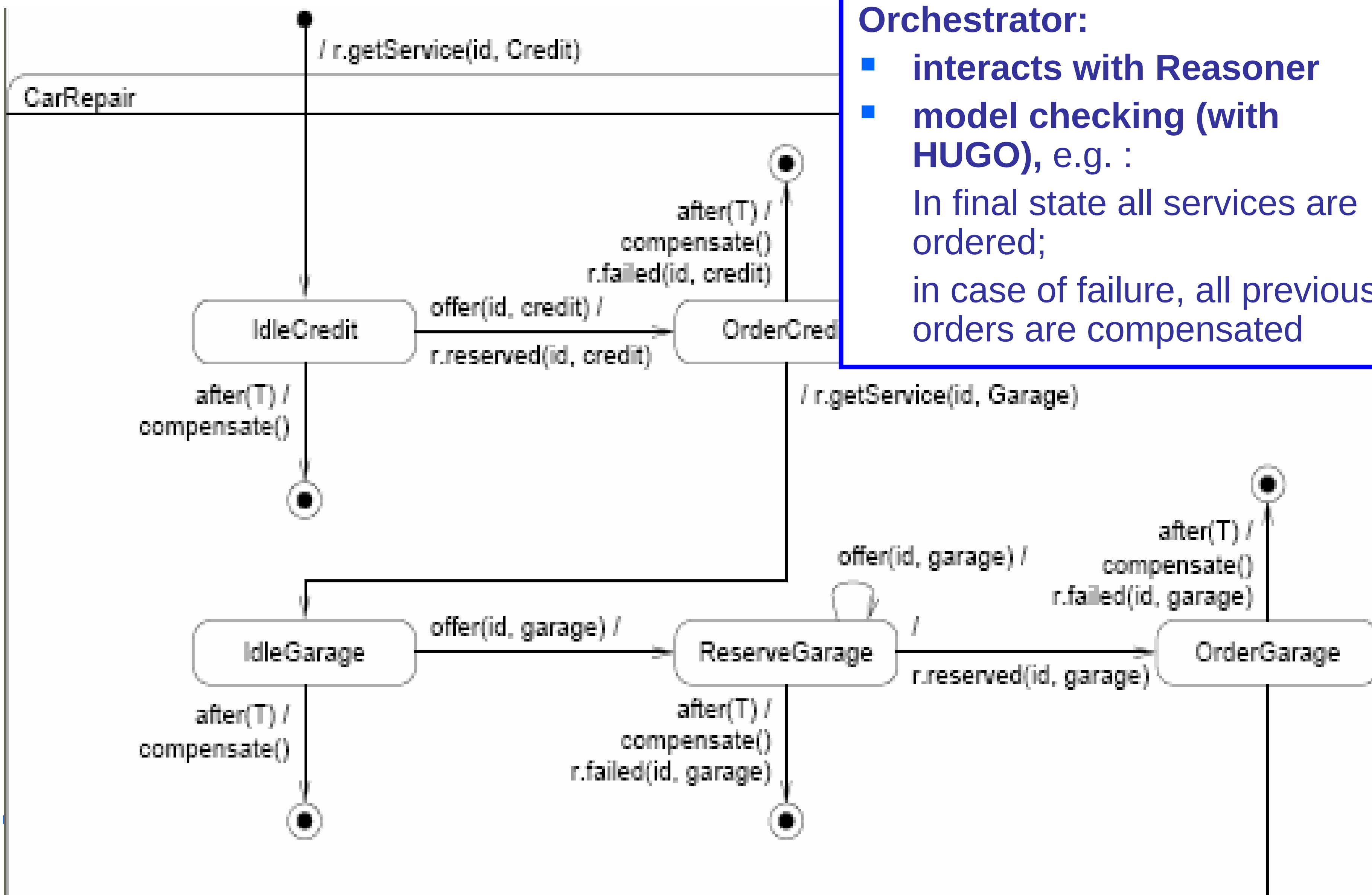
Design and Analysis of Orchestrator

- Specify orchestrator workflow by

  Activity Diagram with Compensation
- **Model Transformation to „classical“ State Diagram** by using car software architecture
- **Quality analysis by model checking** of classical State Diagram
- **Translation to implementation** (currently Java or SystemC)



Model Checking the Orchestrator (with HUGO)



Orchestrator:

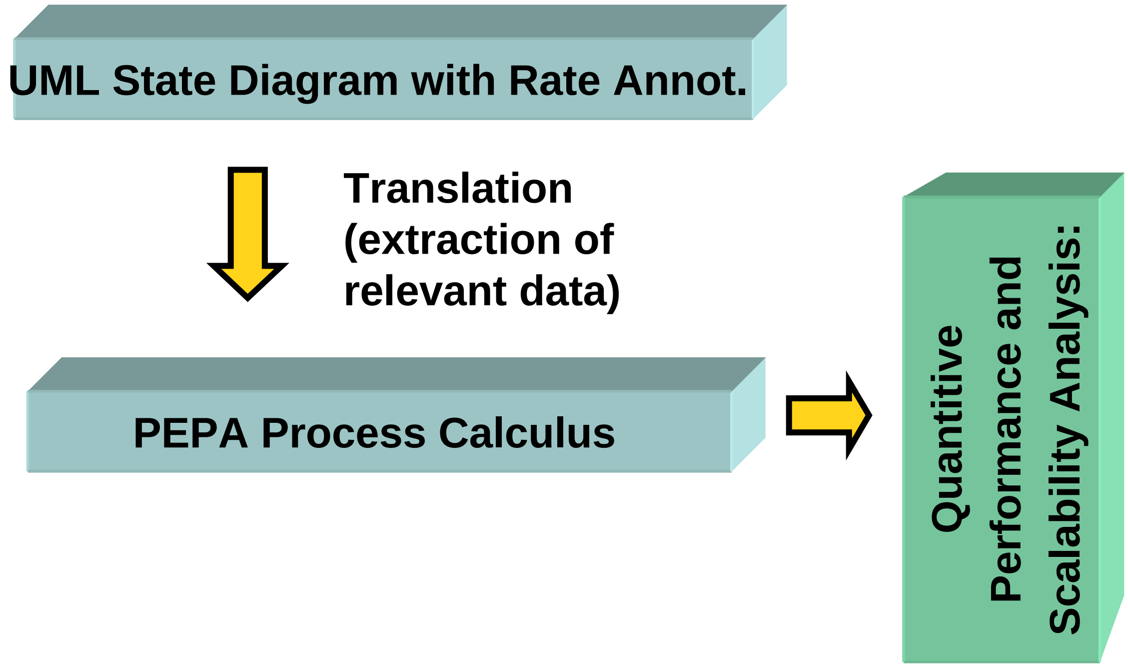
- **interacts with Reasoner**
- **model checking (with HUGO),** e.g. :

  In final state all services are ordered;

  in case of failure, all previous orders are compensated

Analysis of Quantitative Properties: Service Level Agreements

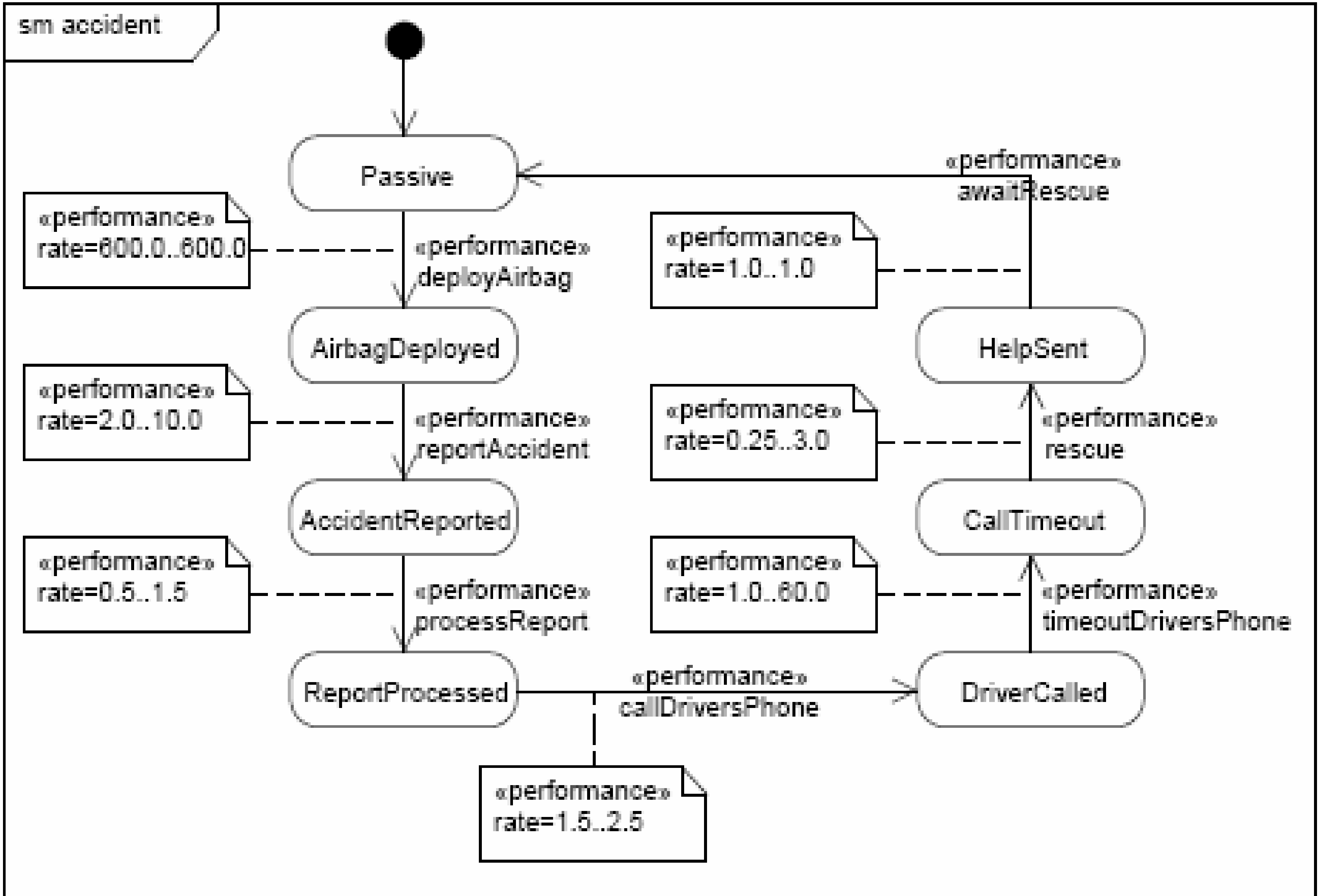
- **Specifying performance by annotating State Machines [DEGAS-Projekt 2004]**
- **Translation into process calculus PEPA [Gilmore 2004]**
- **Performance and scalability analysis of Service Level Agreements with**
  - Continuous Markov chains
  - Ordinary differential equations [Gilmore, Hillston 2005]



Example: Accident Assistance Scenario

- An airbag deploys in 1/10 of a second (Rate: 600)
- The car can transmit location data in 6 to 30 seconds (Rate: 2.0 .. 10.0)
- It takes about one minute to register the incoming data (Rate: 0.5 .. 1.5)
- It takes about thirty seconds to call the driver’s phone (Rate: 1.5 .. 2.5)
- Give the driver from a second to one minute to answer (Rate: 1.0 .. 60.0)
- Vary about one minute to decide to dispatch medical help (Rate: 0.25 .. 3.0)
- The driver is now awaiting rescue.

State Machine with Rate Annotations



Translation to PEPA

- Reporting the accident:

```
Car1 = (airbag, r1).Car2
Car2 = (reportToService, r2).Car3
Car3 = (processReport, r3).Car4
```

- Attempting a dialogue between the service and the registered driver of the car

```
Car4 = (callDriversPhone, r4).Car5
Car5 = (timeoutDriversPhone, r5).Car6
```

- Sending medical help

```
Car6 = (rescue, r6).Car7
```

- And waiting …

```
Car7 = (awaitRescue, r7).Car1
```

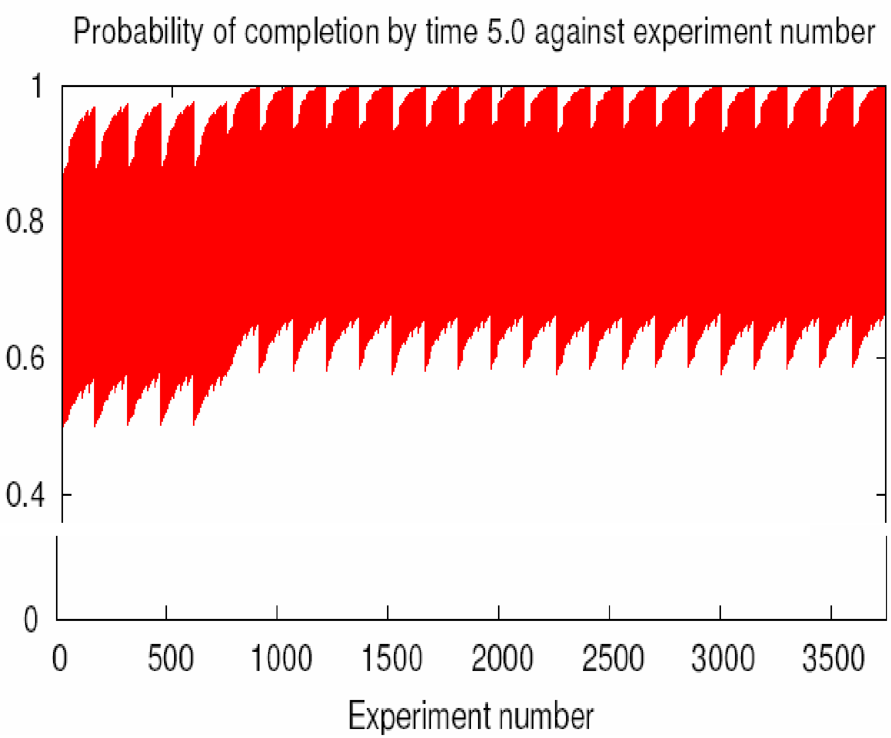
Analysis of Service Level Agreements

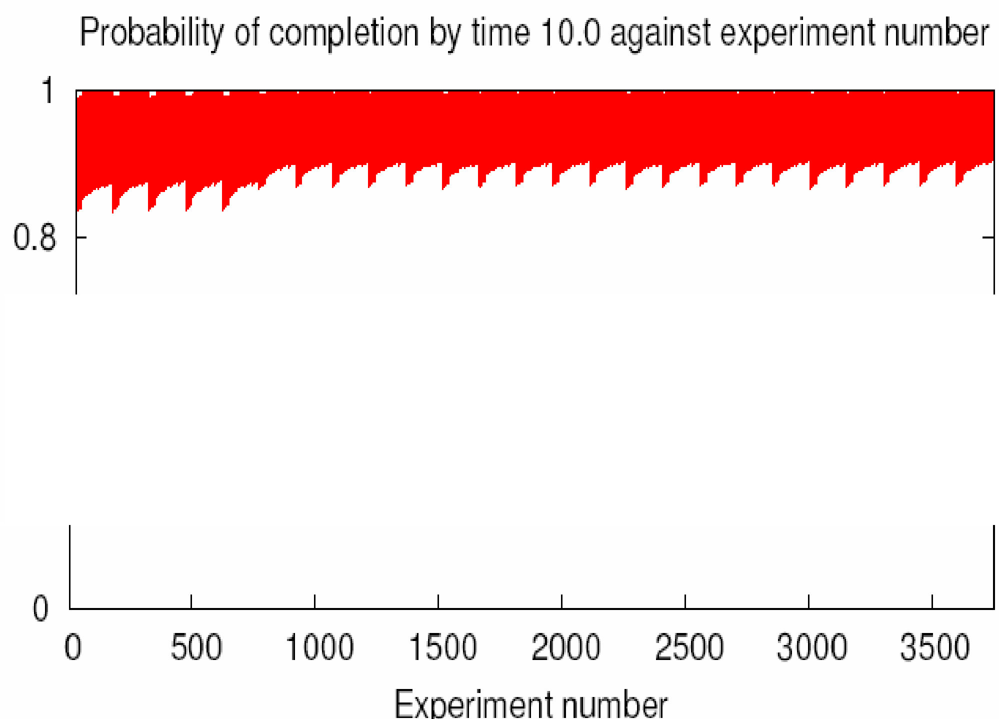
- **Example Service Level Agreement:**

  At least 40% of airbag deployments lead to medical help being sent within five minutes and at least 80% of airbag deployments lead to medical help being sent within ten minutes.

- **Analysis by varying rates r2-r6**:

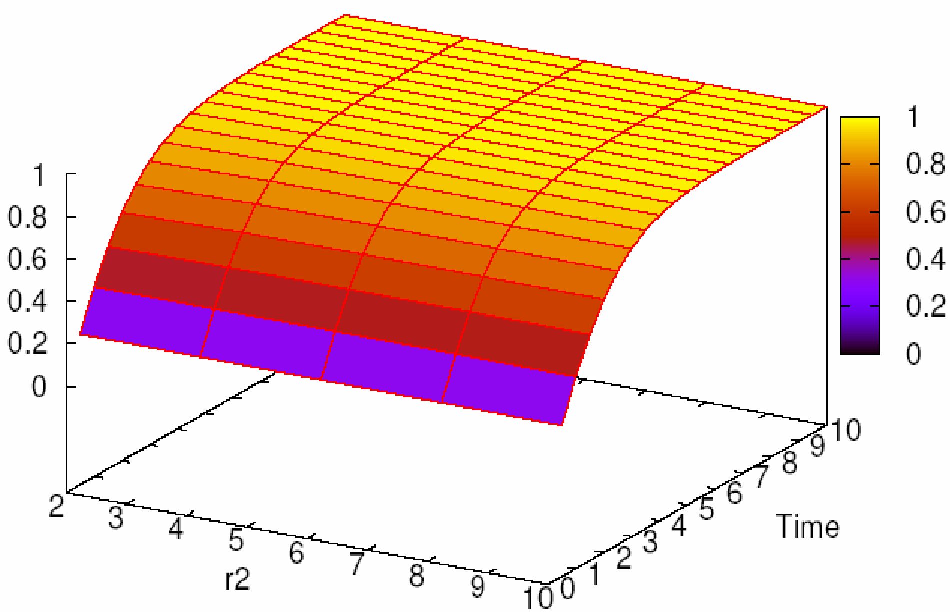
  5 * 5 * 5 * 5 * 6 = experiments with ipc/Hydra Tool [U. Edinburgh]



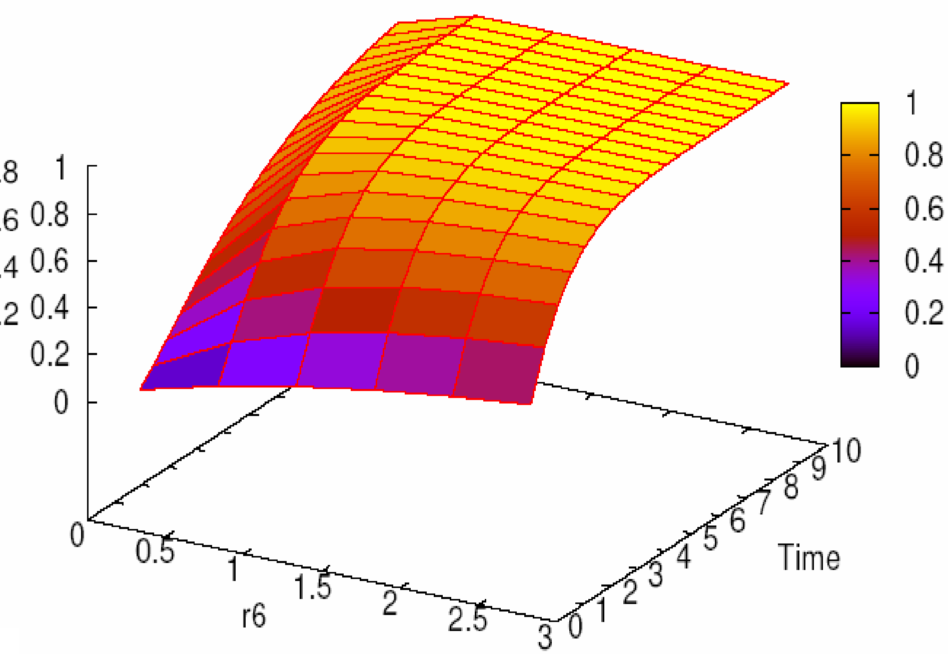


- **Cumulative analysis of Service Level Agreement:**

Sensitivity to variation of r2



Sensitivity to variation of r6



Consequence: A faster decision to dispatch medical help (governed by rate *r*6) is more important than trying to transmit location data faster (governed by rate *r*2),

Concluding Remarks

- **SENSORIA** is developing
  - adequate **linguistic primitives for modelling and programming** global service-oriented systems
    - Phoenix, …, STOKLAIM, …, SRML
  - **qualitative and quantitative analysis methods** for verifying and validating
    - service level agreements, dynamic composition of services, security, trust, resource usage, …
  - **sound engineering and deployment techniques** for global services
    - based on model transformations
- With the goal of building a comprehensive approach for

  Engineering of software systems for
  Service-Oriented Global Computers

  by integrating
  - **foundational theories, techniques, and methods with**
  - **pragmatic software engineering**