IMT 2013
Lucca, Italy
September 2013

Software Engineering and Service-Oriented Systems

Martin Wirsing
LMU München

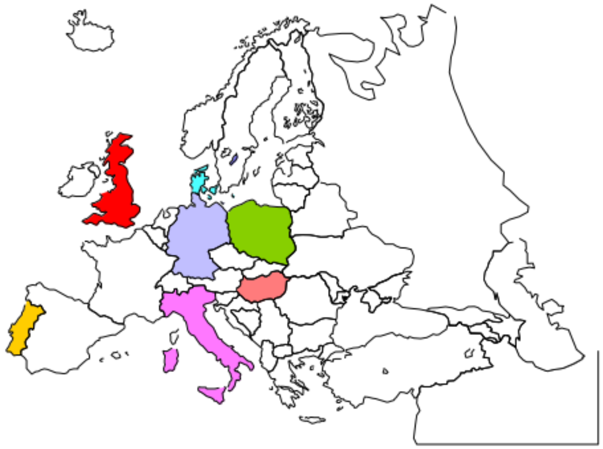
Ludwig-Maximilians-Universität München

in co-operation with Francesco Tiezzi, and
the SENSORIA team, in particular, Nora Koch, Philip Mayer, Rosario Pugliese, Stephen Gilmore and many other SENSORIA members

SENSORIA Project

- EU project of 6th Framework Programme (FP6)
- Information Society Technologies (IST)
- Global Computing (GC2)
- Future and Emerging Technologies (FET)

Consortium



- 19 partners
- 7 countries
- 2005 – 2010
- Coordination: LMU

- LMU Munich (Coordination)
- Universitá di Trento
- University of Leicester
- Warsaw University
- Technical University of Denmark at Lingby
- Universitá di Pisa
- Universitá di Firenze
- Universitá di Bologna
- Istituto di Scienza e Tecnologie della Informazione
- University of Lisbon
- University of Edinburgh
- ATX Software SA
- Telecom Italia S.p.A.
- Imperial College London
- University College London
- Cirquent GmbH
- Budapest University of Technology and Economics
- S&N AG
- School of Management of Politecnico di Milano

Contents

1. SENSORIA Project Overview and Results
2. Model-Driven Development of Service-Oriented Systems
3. Modal I/O Transition Systems as Semantics of UML4SOA
4. Summary: SENSORIA: Software Engineering and Service-Oriented Systems
5. Introduction to Modeling and Developing “Ensembles” with ASCENS

Contents

1. Project Overview and Results
   - Service-oriented computing
   - The ***SENSORIA*** project
   - Technical results
   - Further results
   - SENSORIA in numbers

Service-oriented computing

- Service-Oriented Computing (SOC)
  - the compute paradigm behind service-oriented systems, i.e. for organizing and utilizing distributed capabilities that may be under the control of different ownership domains
- Service-Oriented Architecture (SOA)
  - an architectural style to realize SOC
  - promise to organize and understand organizations, communities and systems maximizing agility, scalability and interoperability
  - very often built by IT industry in an ad-hoc and undisciplined way

Setting the scene
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
    - like gas, power, telephone, etc.

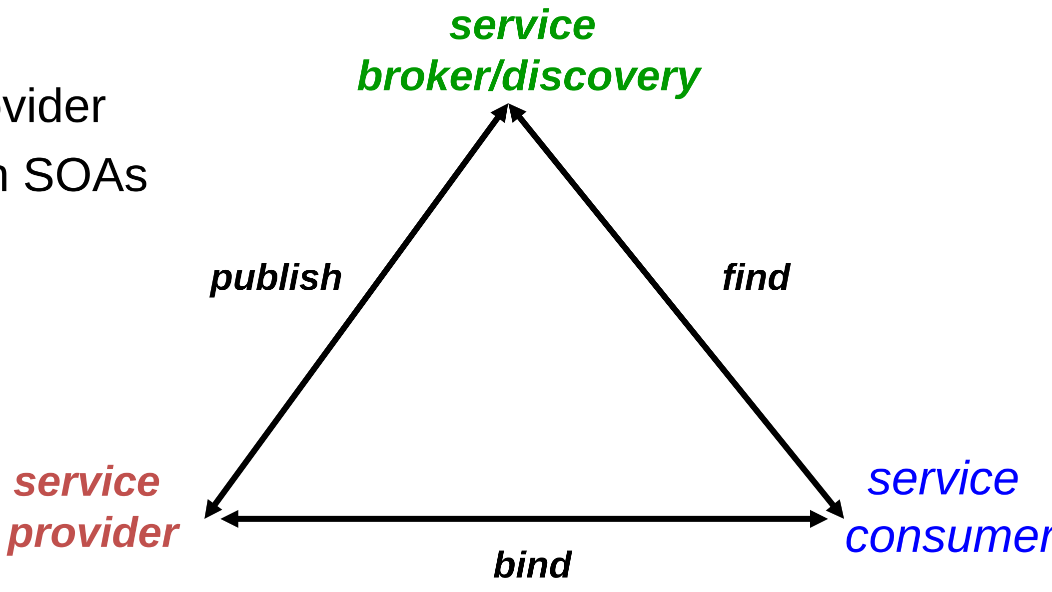
- Service-Oriented Systems (SOS)
  - use loosely coupled services
  - massively distributed, interoperable, evolvable applications
  - consist of providing, consuming and publishing services, i.e. establishing a community or marketplace
    - like applications spread over the web, e.g. online banking, hotel reservation, flight booking, etc.

Software engineering for SOS
(Service engineering)

- Challenges for service engineering
  - specification and querying services
  - correctness and consistency
  - automated composition of services (orchestration) guaranteeing availability and reliability
  - compensation of long running transactions
  - evaluating and implementing sustained performance, security and safety, adaptive behaviour, …
  - deployment and re-engineering

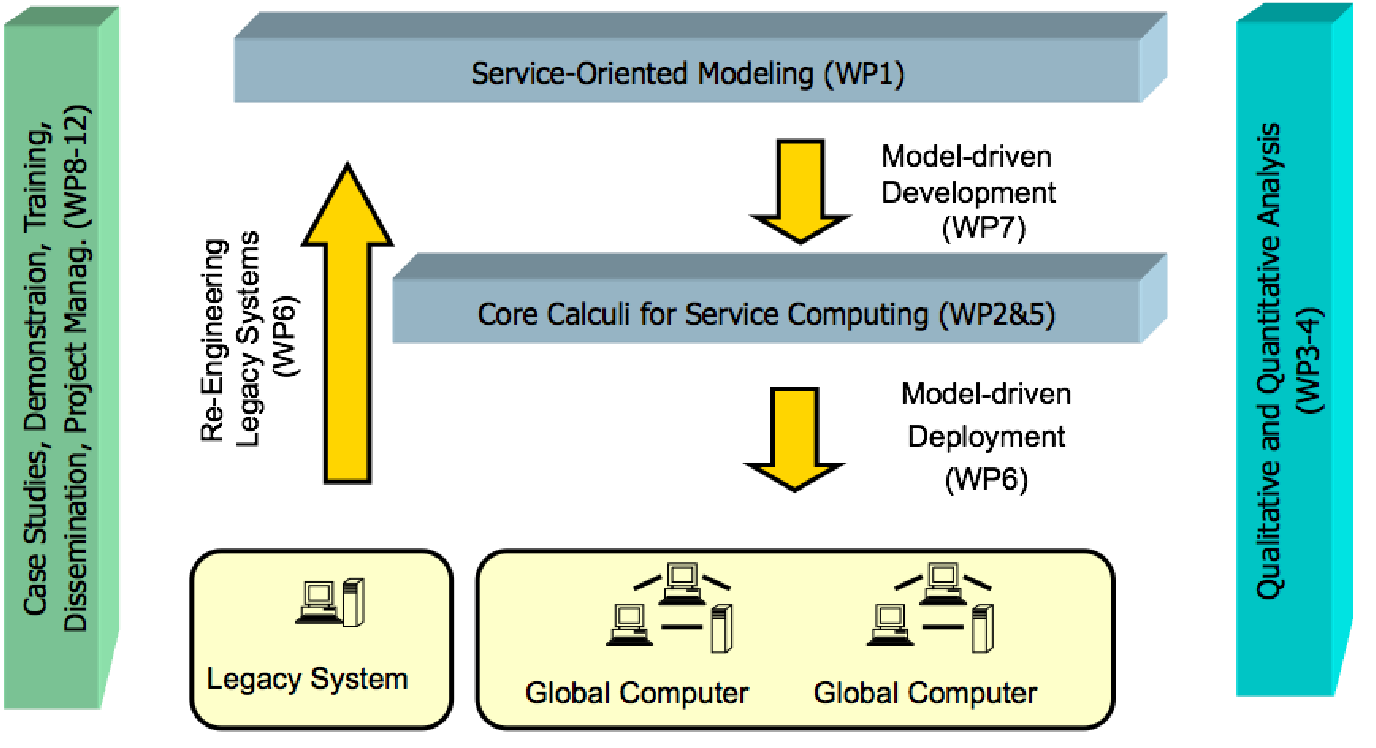
Stakeholders/Parties in SOAs

- Service providers
  - offer services that correspond to ‘market’ demands
- Service consumers/requesters
  - are applications, not people
  - are decoupled from the providers
  - binding to services at run time, not design time
- Service brokers
  - manage registries
  - binds consumer and provider
  - offered as middleware in SOAs
- SOA triangle



SENSORIA approach

- Rigorous comprehensive approach to engineering service-oriented systems
- Integration of
  - foundational theories, techniques, and methods
  - pragmatic software engineering



… more details

- Modelling front-end

  Service-oriented applications are designed using high-level visual formalisms such as the industry standard UML or domain-specific modelling languages.

- Hidden formal analysis of services

  Back-end mathematical model analysis is used to reveal performance bottlenecks, or interactions leading to errors or violation of service contracts.

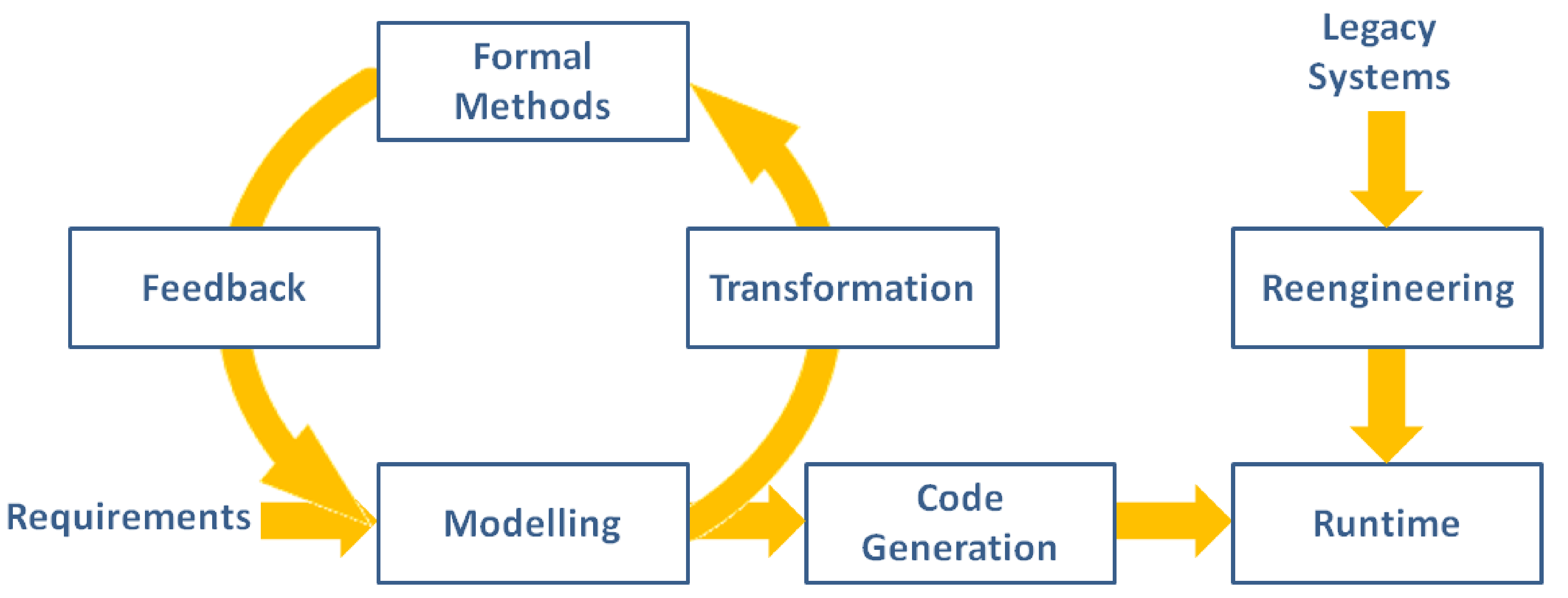
- Automated model transformations

  Formal representations are generated by automated model transformations from engineering models.

- Service deployment

  As a result, service models of proven quality serve as the basis for deployment transformations to generate configurations for standards-compliant platforms.

Model of the SENSORIA model-driven development approach



SENSORIA results

- Languages
- Techniques
- Methods
- Tools

to support this development process and the analysis of service-oriented systems

Result topics

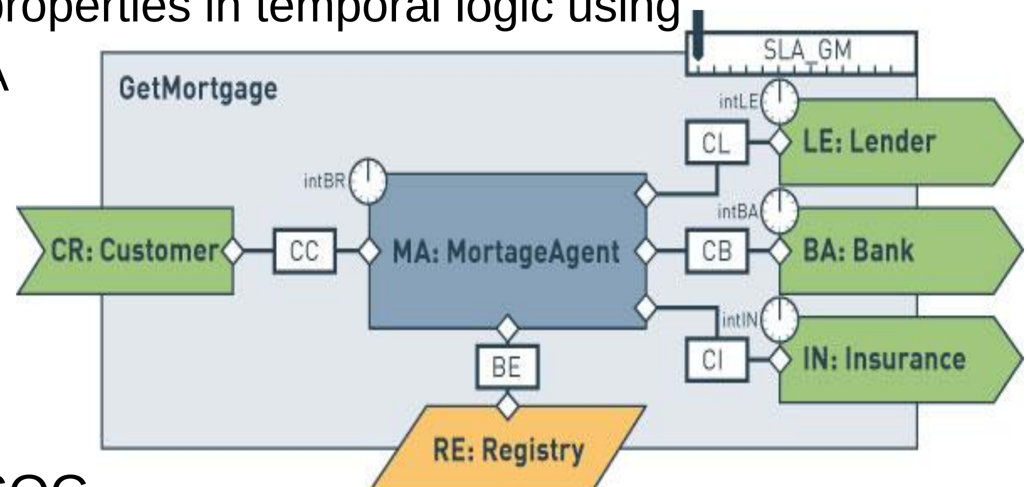
- 3 research themes
  - language primitives for global service-oriented computing
  - qualitative and quantitative analysis methods for
  - sound engineering methods and deployment techniques

- complemented by
  - case studies
  - dissemination
  - demonstration and training
  - exploitation

a few words on the most important results

Language primitives

- SRML
  - declarative high-level language for service-oriented systems
  - layer static and dynamic service composition
  - reasoning about system properties in temporal logic using UCTL/UMC, SRMC/PEPA
  - well-defined mathem. semantics, editor



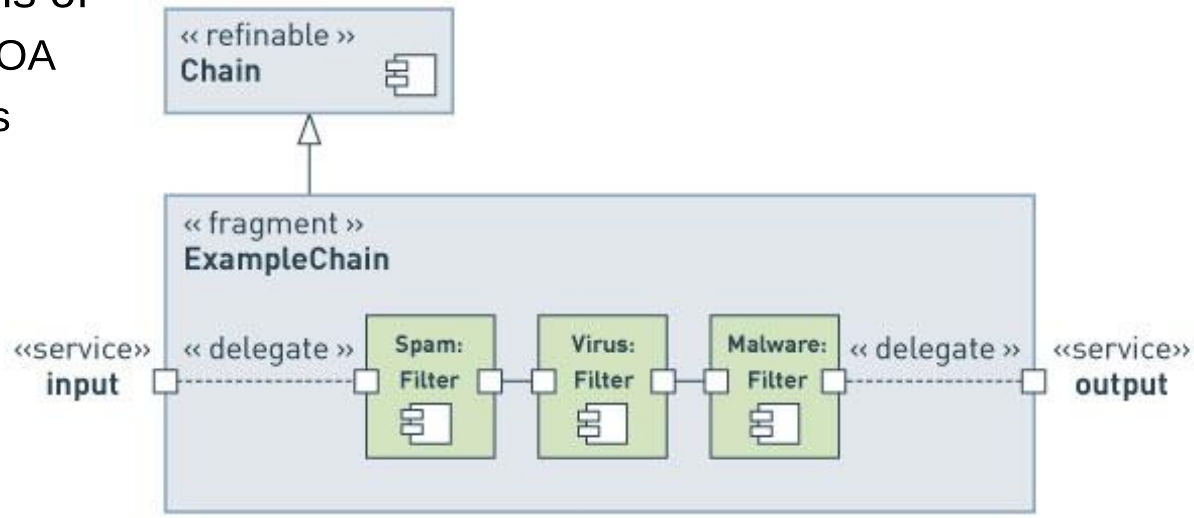
- UML family of profiles for SOC
  - orchestration of services
  - service-level agreements
  - non-functional properties of services
  - implementation of service modes and service deployment

Language primitives (cont.)

- Process calculi for services
  - core calculi needed
    - to describe, discover and compose systems
    - to prove that their behaviour is consistent with the expectation of the designer
  - type inference for session-types, structured patterns of communication
  - extension of local policies mechanisms in order to manage resources
- JOLIE
  - Process calculus based programming language for designing, developing and deploying services and orchestrations

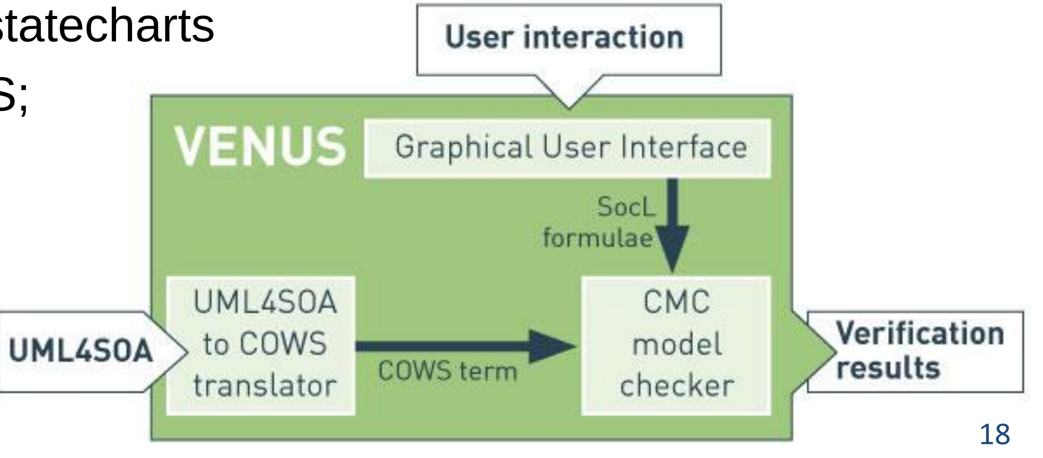
Language primitives (cont. 2)

- Composition of services
  - full integration of SLA primitives with transaction primitives
  - assessment of theories and techniques for choreography conformance
  - formal comparison of
    - long running transactions and
    - compensations.
  - ADR formalizations of
    - SRML, UML4SOA
    - software modes



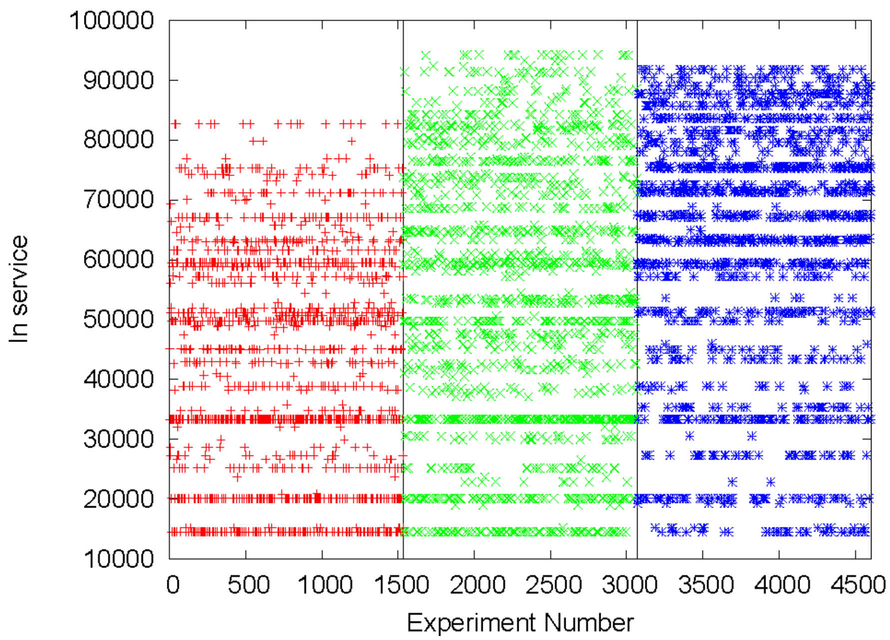
Qualitative analysis methods

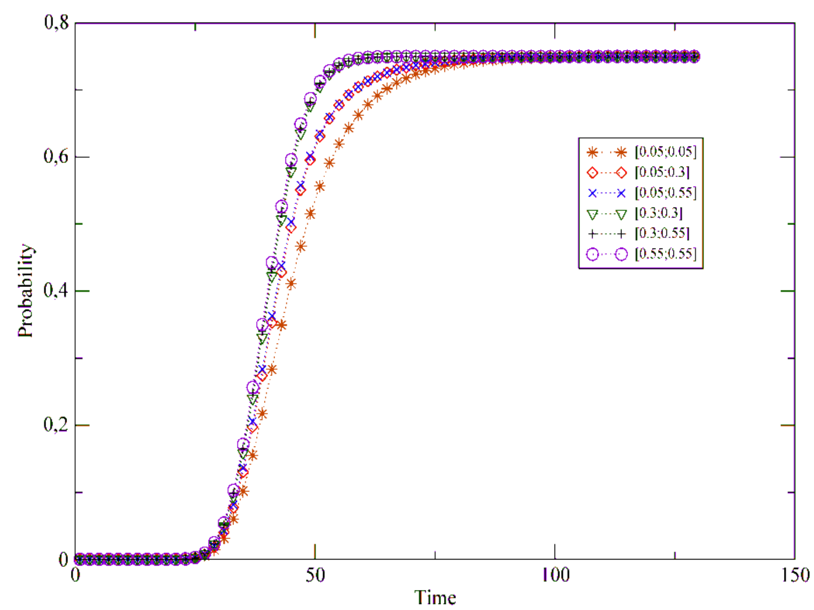
- Adaptive and Dynamic Service Compositions
  - LTSA WS-Engineer+ Modes tool provides mechanical support for the analysis of Service Mode models
  - to ensure safety and correctness of adaptive and dynamic service composition specifications
- BPEL Analysis and Back-Annotation.
  - end-to-end method which facilitates analysis of several liveness and safety properties of BPEL orchestrations
- CMC/UMC and Venus
  - two prototypical modelcheckers for analysing qualitative properties
  - UMC based on UML statecharts
  - CMC based on COWS;
  - Venus System



Quantitative analysis methods

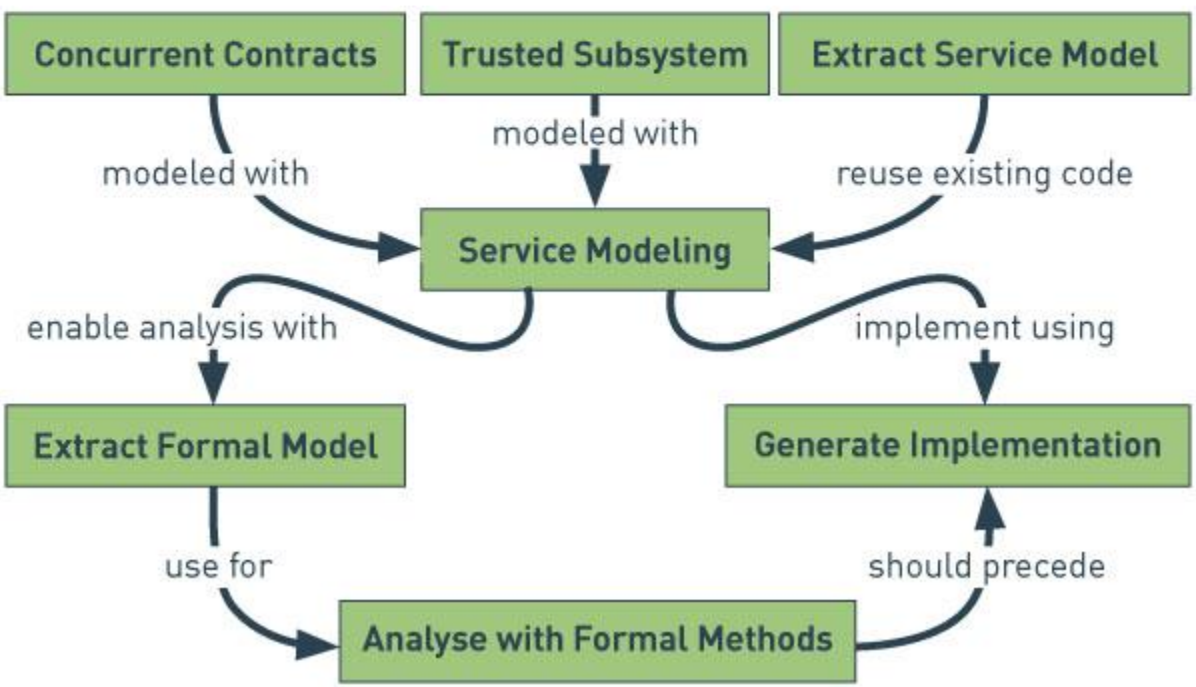
- **Model checking stochastic calculi for services**
  - model checking MarCaspis vs. SoSL formulae
  - model checking sCOWS with sCOWS-LTS and sCOWS-AMC
- SRMC - SENSORIA Reference Markovian Calculus
  - stochastic process calculus which captures inherent uncertainty in SOSs
  - allows the model to express both kinds of uncertainty and to evaluate this to give performance predictions which are valid whichever configuration of service providers is selected





Sound engineering methods

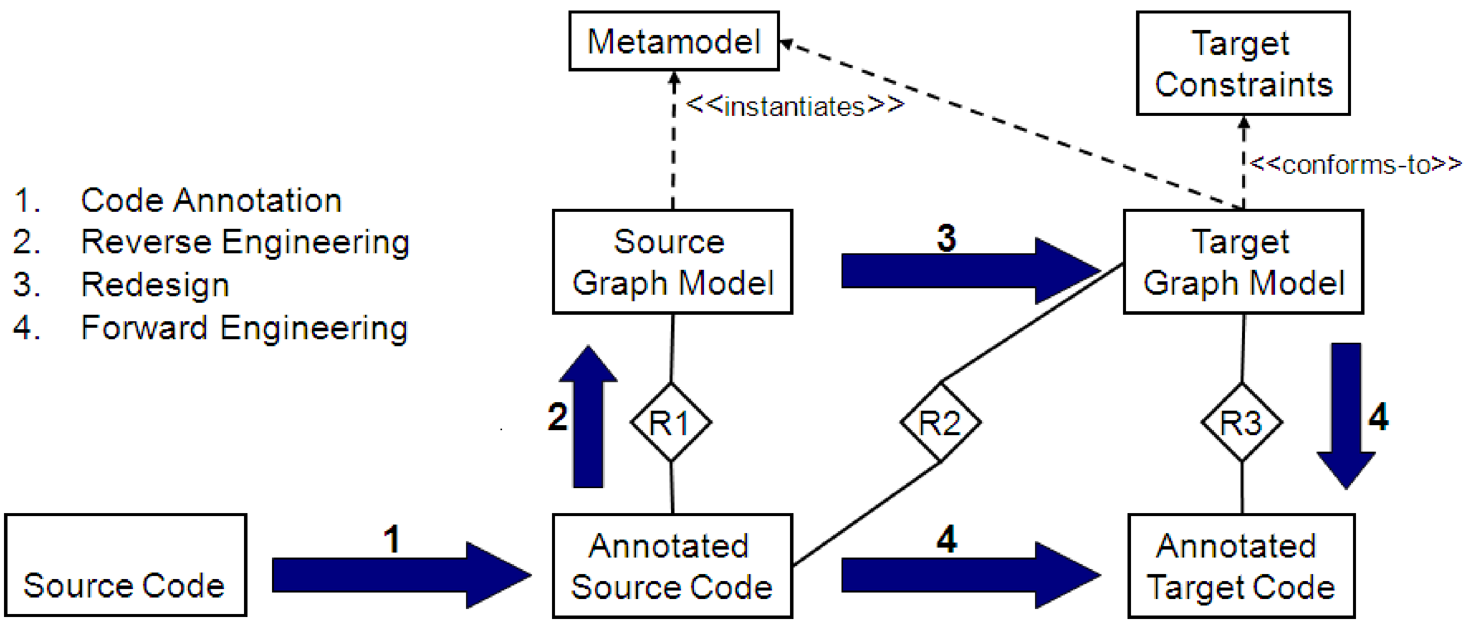
- Engineering
  - Eclipse-based SENSORIA development environment (SDE)
  - model-driven transformations for deployment, supporting WS-Security & WS-Reliable Messaging
  - WS-Engineer & natural language-based analysis tool VENUS
  - Performance modelling with SRMC
  - pattern-based approach



Reengineering and deployment techniques

- Deployment techniques
  - end-to-end support for dynamic service composition from modelling to runtime
  - deployment and brokering with Dino
- Re-engineering
  - prototype for re-engineering legacy applications to SOA

1. Code Annotation
2. Reverse Engineering
3. Redesign
4. Forward Engineering



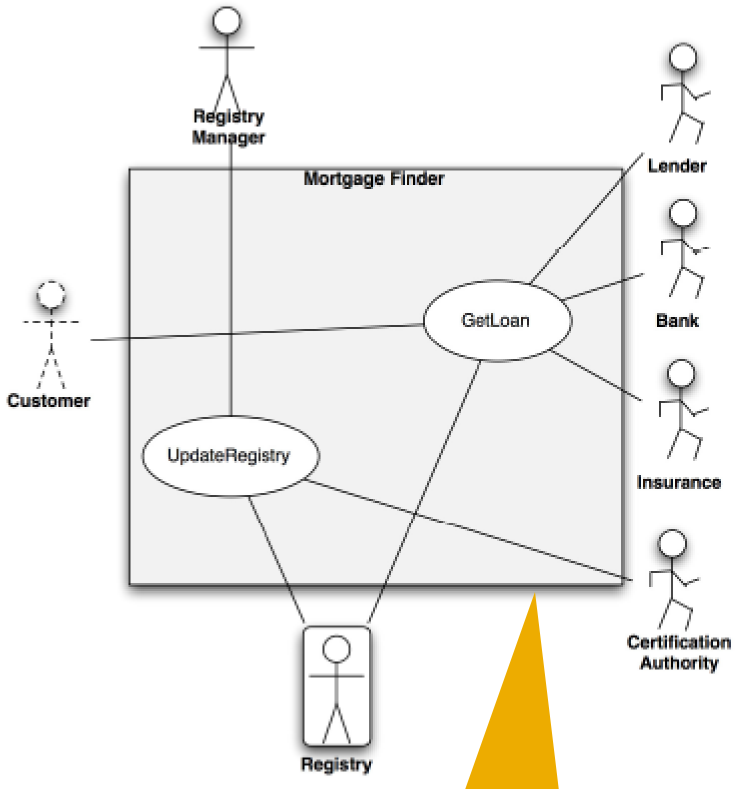
… concrete results

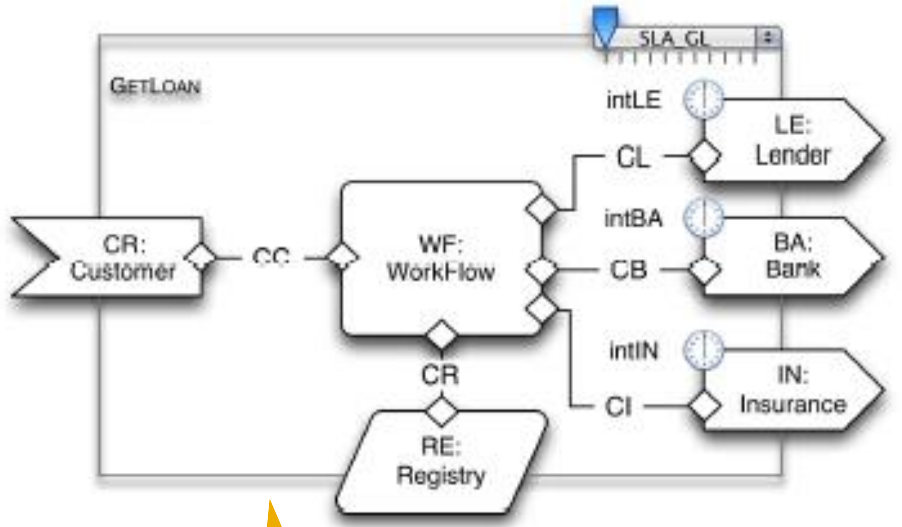
- Service ontology
- Modelling languages
  - UML4SOA, SRML, StPowla
- Process calculi
  - COWS, SCC, SOCK, Stock, …
- Languages for programming service-oriented systems
  - Jolie
- Transformation tools supporting MDE process
  - SRML Use Case Wizard
    - UseCases2SRML
  - MDD4SOA
    - UML2BPEL/WSDL, UML2Jolie, UML2Java
    - BPEL/WSDL transformers (ActiveBPEL, Tomcat)
  - VIATRA
    - SOA2WSDL, UML2Axis

… concrete results (cont.)

- Languages, tools and techniques for qualitative and quantitative analysis
  - SRMC/PEPA, WS-Engineer, Venus/CMC/UMC, Lysa, StockKlaim, MoSL
- Service broker
  - Dino
- Re-engineering tool
  - CareStudio
- CASE tool
  - SRML modelling environment
- Tool suite
  - SENSORIA Development Environment (SDE)

Example: From use cases to SRML



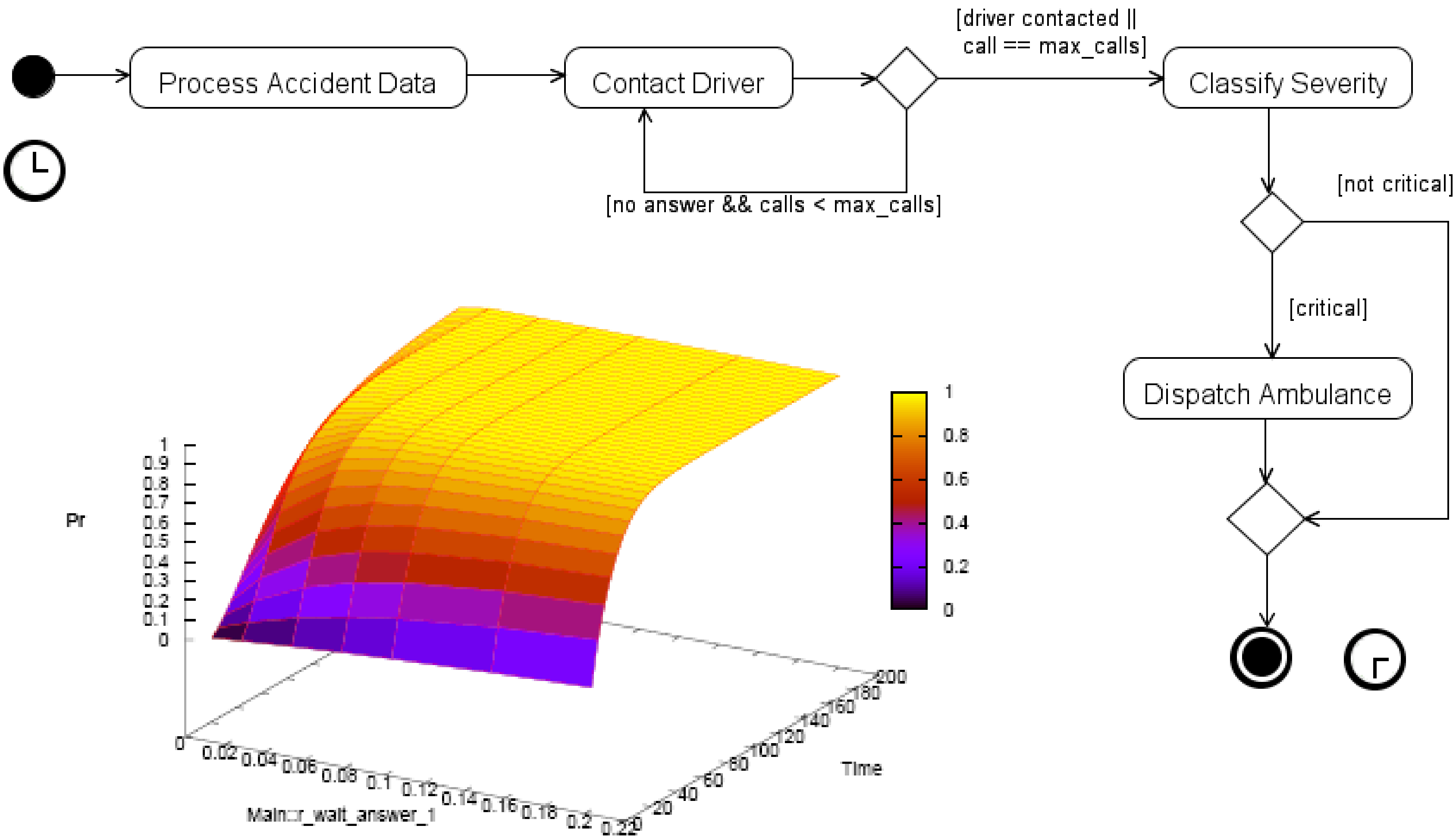


Service-oriented use case diagram

Derived SRML module for GetLoan

Example: Quantitative analysis with UML

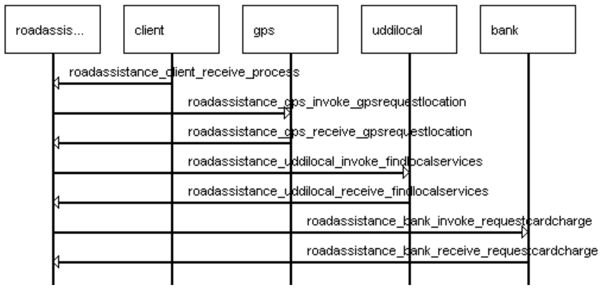
Accident scenario of automotive case study



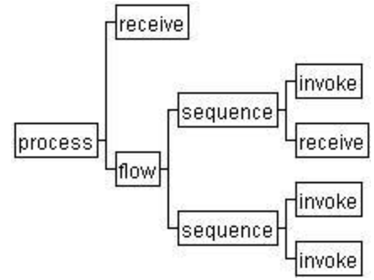
Example: Qualitative analysis approach

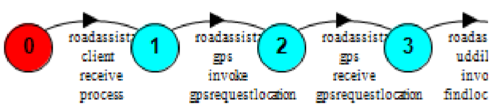
Safety: Design vs. Implementation

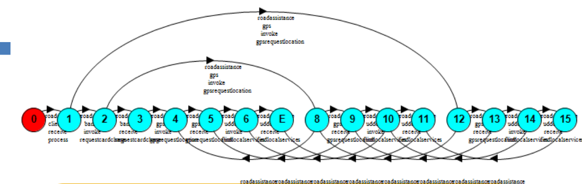
MSC



WS-BPEL

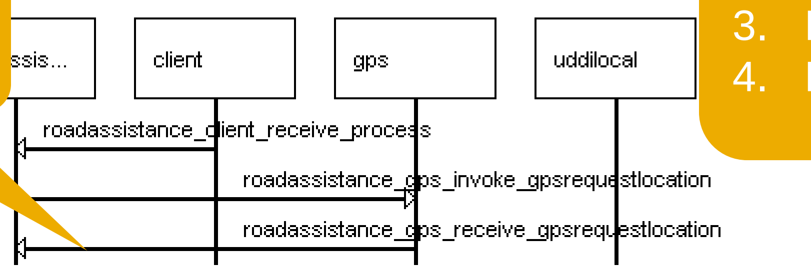






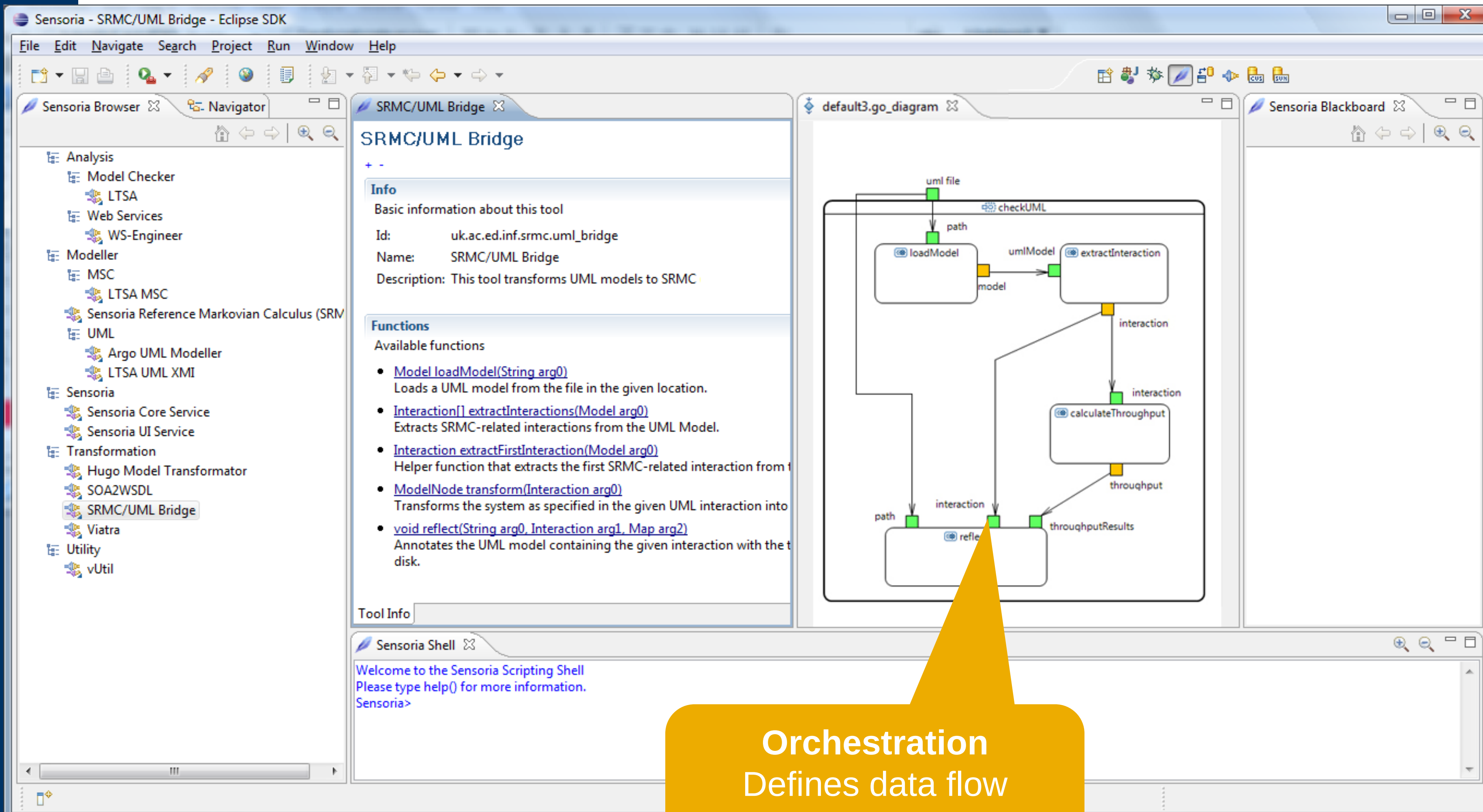
1. Mappings
2. Compilation of LTS
3. Properties
4. Reachability Search

Implementation does not fulfil scenario X



Example: SDE user interface

Graphical orchestration of tools



Orchestration
Defines data flow between tool functions

See later

Examples: Security protocol

1. SA-TEK-Challenge

BS → MS: $BS\_Random, KeySeqNo, AKID, [KeyLifeTime], H-C/MAC$

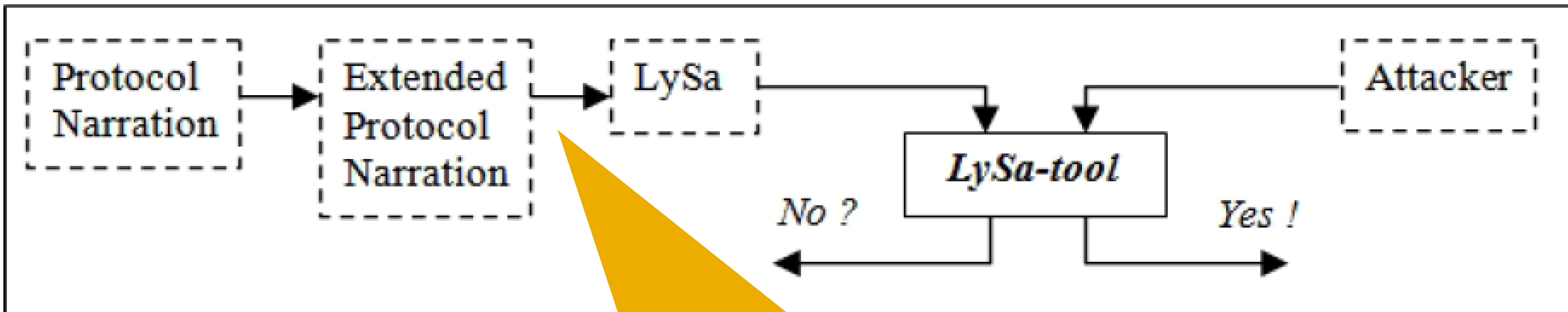
2. SA-TEK-Request

MS → BS: $MS\_Random, BS\_Random, KeySeqNo, AKID, SecurityCapabilities, SecNegParam, PKMConfSettings, H-C/MAC$

3. SA-TEK-Response

BS → MS: $MS\_Random, BS\_Random, KeySeqNo, AKID, [SA-TEKUpdate], FrameNo, [SADescriptors], SecNegParam, H-C/MAC$

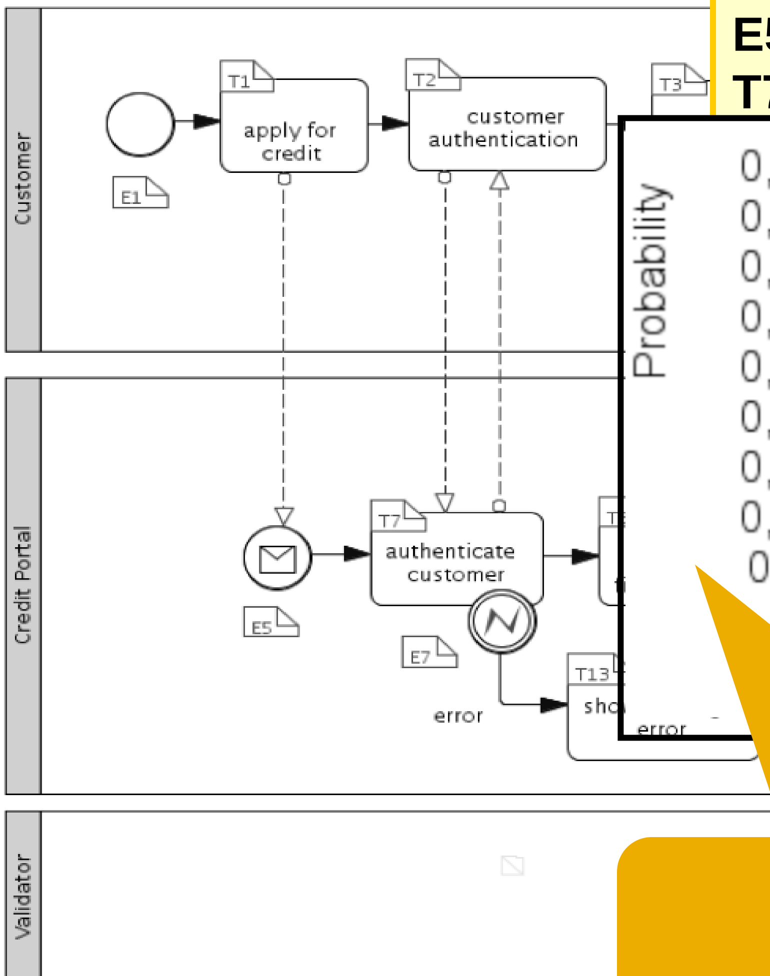
$$
\begin{aligned}
&(\nu\, K)\,(\nu\, id)\,(\\
&!\ (\nu\, na)\,\langle id, na, \{\{\!|id, na|\!\}_{Hash}\}_K[\text{at } a1 \text{ dest } \{b1\}]\rangle.\\
&\quad (na, id;\ xnb, xS, xmac).\\
&\quad \text{decrypt } xmac \text{ as } \{\{\!|na, id, xnb, xS|\!\}_{Hash};\ \}_K[\text{at } a2 \text{ orig } \{b2\}] \text{ in}\\
&\quad (\nu\, T)\,\langle na, nb, id, T, \{\{\!|na, nb, id, T|\!\}_{Hash}\}_K[\text{at } a3 \text{ dest } \{b3\}]\rangle.0\\
&|\\
&!\ (id;\ yna, ymac).\\
&\quad \text{decrypt } ymac \text{ as } \{\{\!|id, yna|\!\}_{Hash};\ \}_K[\text{at } b1 \text{ orig } \{a1\}] \text{ in}\\
&\quad (\nu\, nb)\,(\nu\, S)\,\langle yna, id, nb, S, \{\{\!|yna, id, nb, S|\!\}_{Hash}\}_K[\text{at } b2 \text{ dest } \{a2\}]\rangle.
\end{aligned}
$$



Verifying and simplifying the PKMv2 Protocol of the credit request scenario [Yuksel, Nielson et al. 07]

Example: Quantitative analysis with BPMN

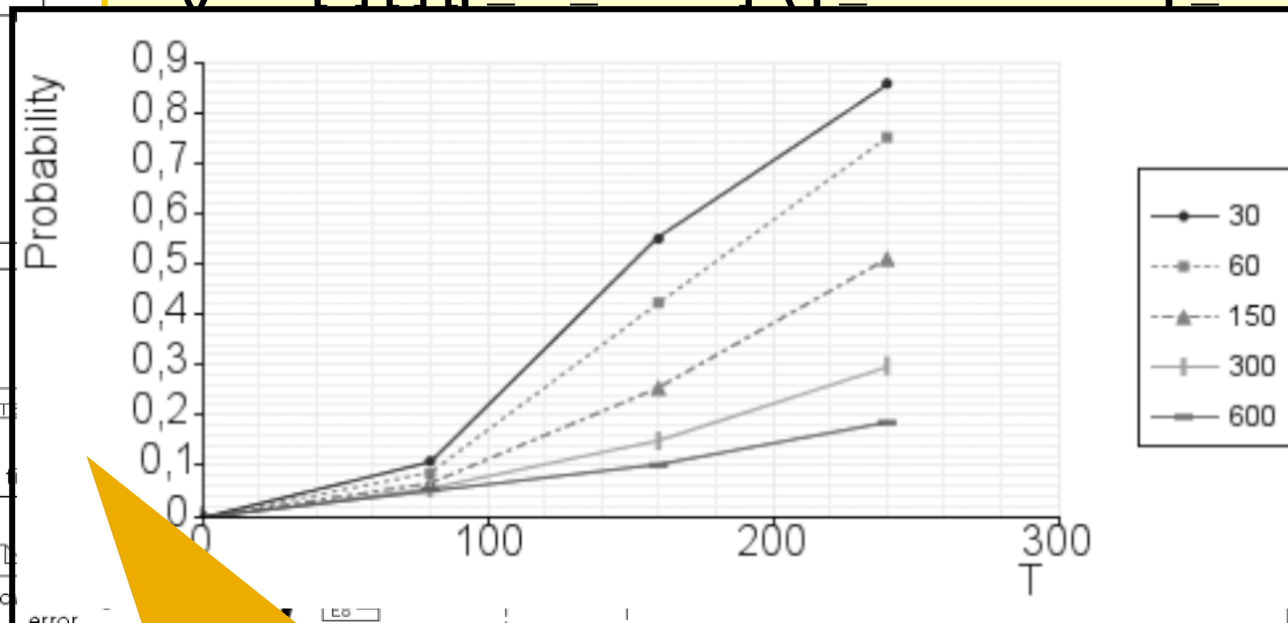
Credit request scenario of the finance case study



-- CREDIT PORTAL --

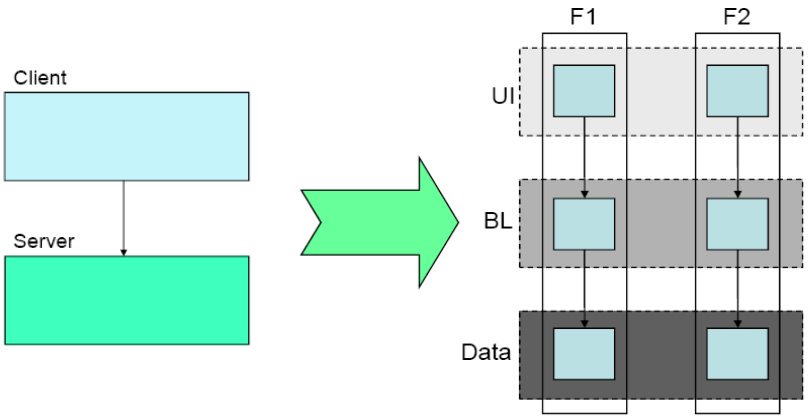
E5() = * p_T1E5 .o?<> . p_E5T7 .o!<>

T7() = * [k] [l] [p_T7_done#] (p_E5T7 .o?<> . p_T2T

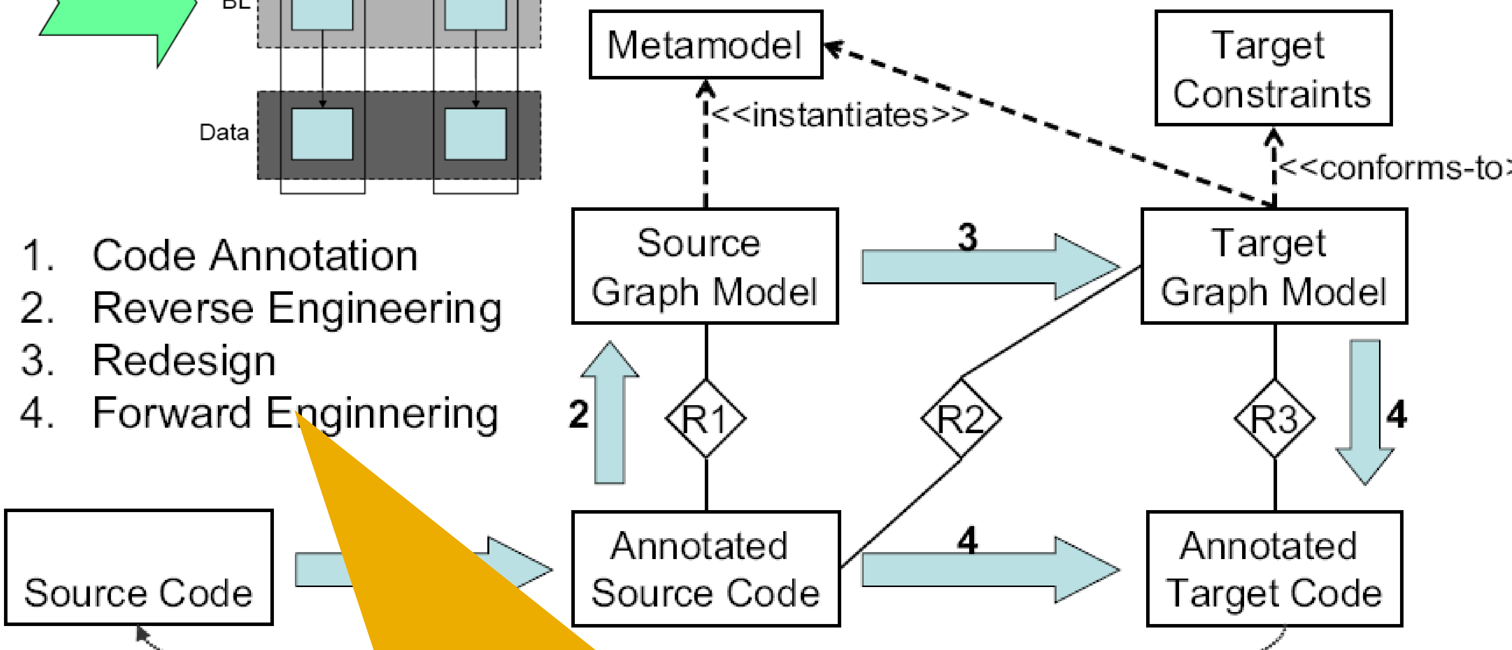


P = ? [true U[T,T] terminate]
Which is the probability that
the system terminates at time T?

Example: Reengineering



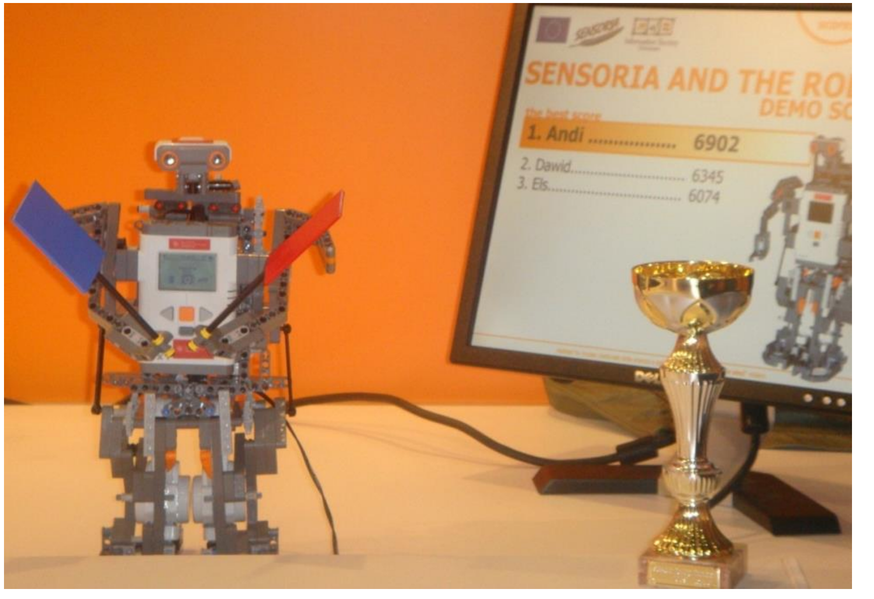
1. Code Annotation
2. Reverse Engineering
3. Redesign
4. Forward Enginnering



Methodology for transformation-based reengineering: client-server to 3-tier SOA

Case studies

- Finance
- Automotive
- Telecommunications
- eUniversity
- Robot bowling
  - ICT 2008
  - FET 2009



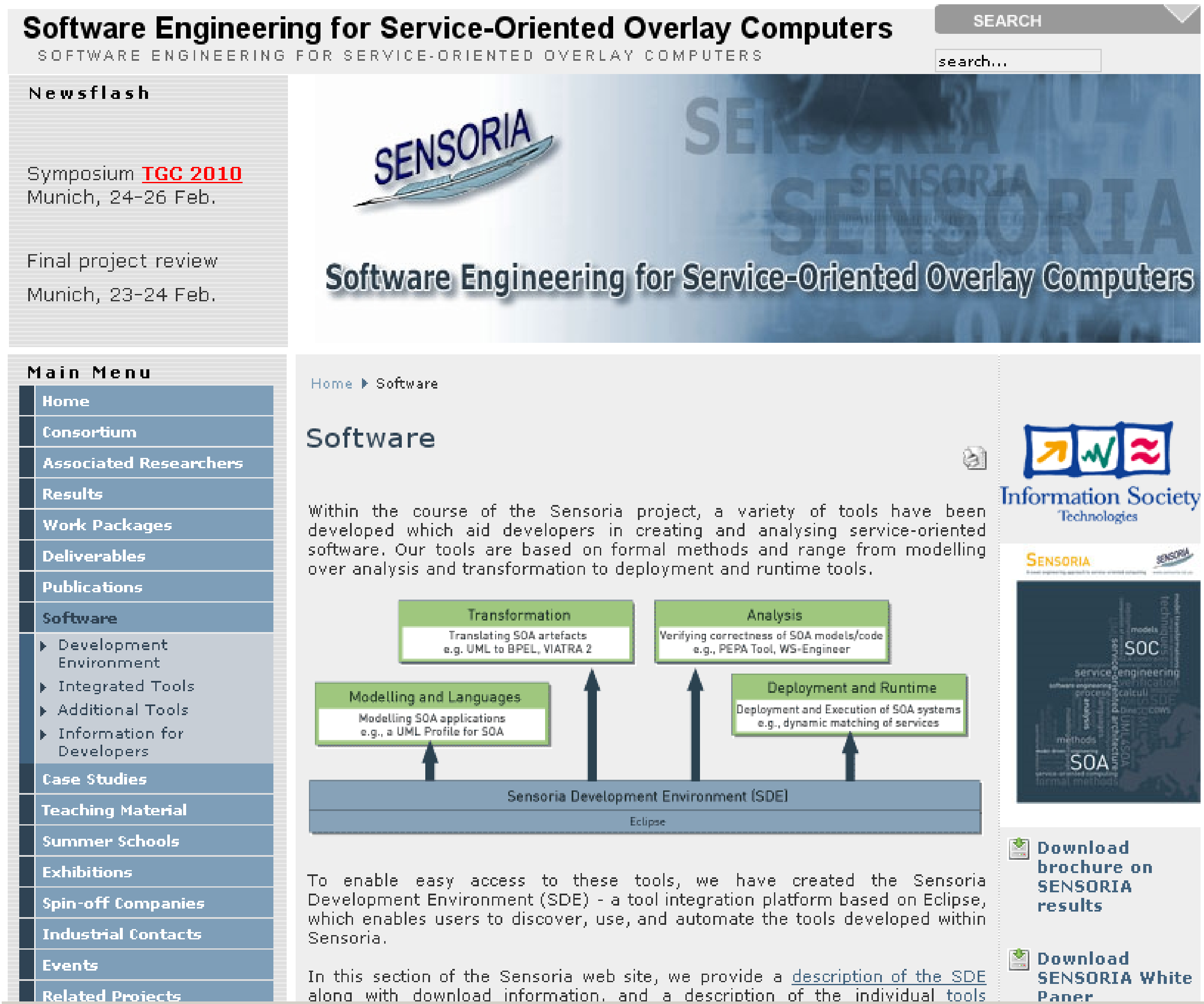
- Modelling case studies
  - SRML, UML4SOA, COWS, SOCK, (Mar)CaSPiS, λ^{req}, CC
- Analysing case studies
  - CMC/UMC, Venus, LySA, WS-Engineer , ChorSLMC
  - SoSL, SRMC/PEPA, λ^{req}, sCOWS-lts
- Model-driven development of case studies
  - SDE, MDD4SOA, service pattern, modes/Dino

Further results: Spin-off companies

- AGILOGIK
  - 2009
  - monoidal soft constraint solver for optimization problems
  - Steingaden, Germany
- Italiana Software
  - 2007
  - design and implementation of SOAs with Jolie
  - Imola, Italy
- OptXware
  - 2005
  - model transformations with VIATRA2
  - Budapest, Hungary

SENSORIA website

www.sensoria-ist.eu



SENSORIA in numbers

Publications	652
- book chapters	16
- articles in journal	139
- papers in conferences and workshop	402
Presentations and tutorials	**192**
PhD Thesis on SENSORIA results	
- finished	29
- ongoing	24
Courses on project results	108
Organization of conf. and workshops	126
Summer schools	3
Fairs and exhibitions	4
Software	
- SDE	1
- integrated tools	19
- additional tools	8

2. MDD4SOA – Model-Driven Development of Service-Oriented Systems

Martin Wirsing
LMU München

Nora Koch
LMU München and Cirquent GbmH

Ludwig Maximilians Universität München

in co-operation with the SENSORIA team

Aim of Chapter 2.

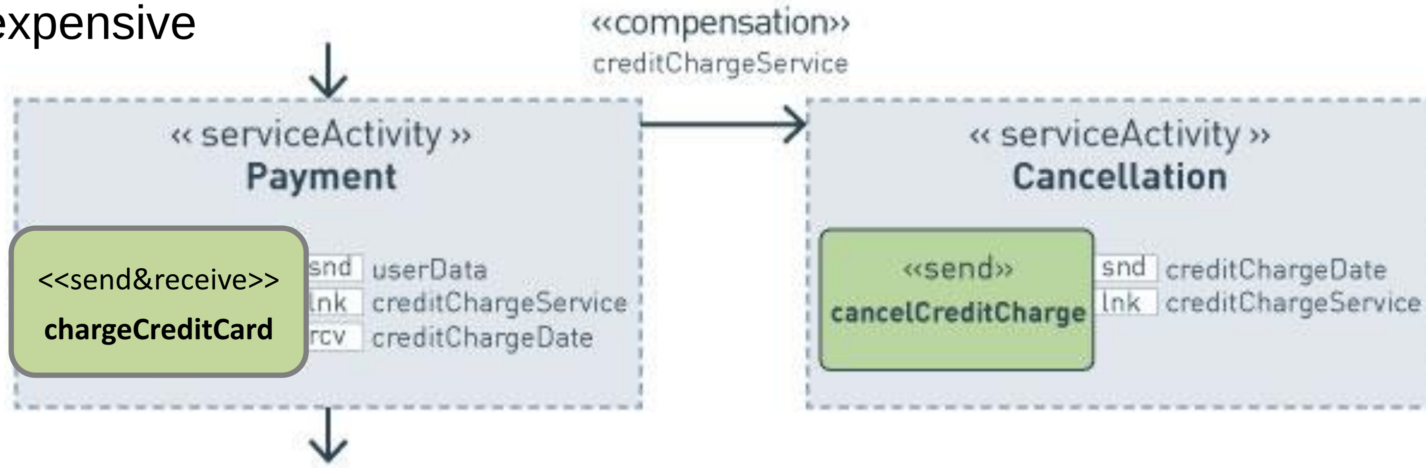
- to provide you with an overview to a model-driven development approach for service-oriented systems that we developed in the SENSORIA project
  - methodological aspects of the engineering process
  - a modelling language
  - a model-driven development environment

Plan of Chapter 2.

- Models and model-driven development
- Modelling
  - Business models
  - Design models
  - Metamodel and model transformations
  - Technical specification
- Model-driven development @ work
  - Tool support by SDE
  - Pattern language
  - Case study

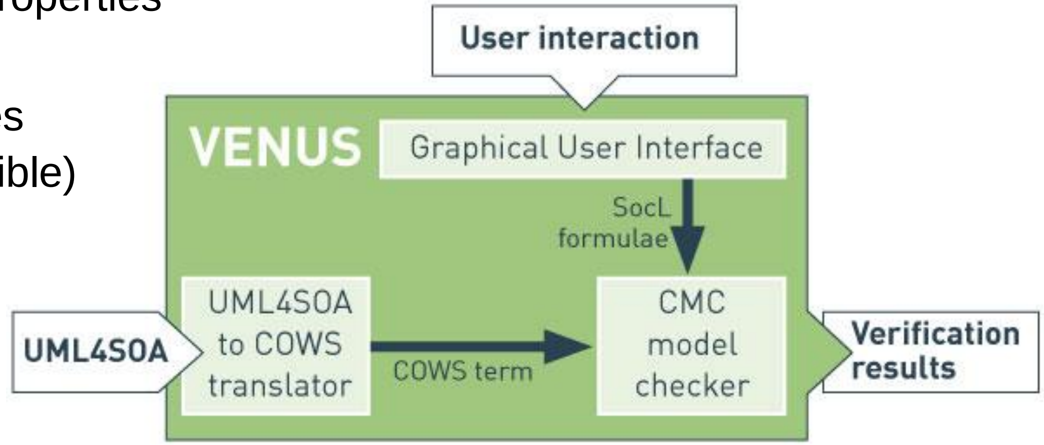
Models in SENSORIA

- A model is used to *describe or specify* SOSs for some certain *purpose*. A model is often presented as a combination of drawings and text. [according definition of MDA Guide, 2003]
- Characteristics models should fulfil [Selic,IEEE,2003]
  - abstract
  - understandable
  - accurate
  - predictive
  - inexpensive



Use of models in SENSORIA

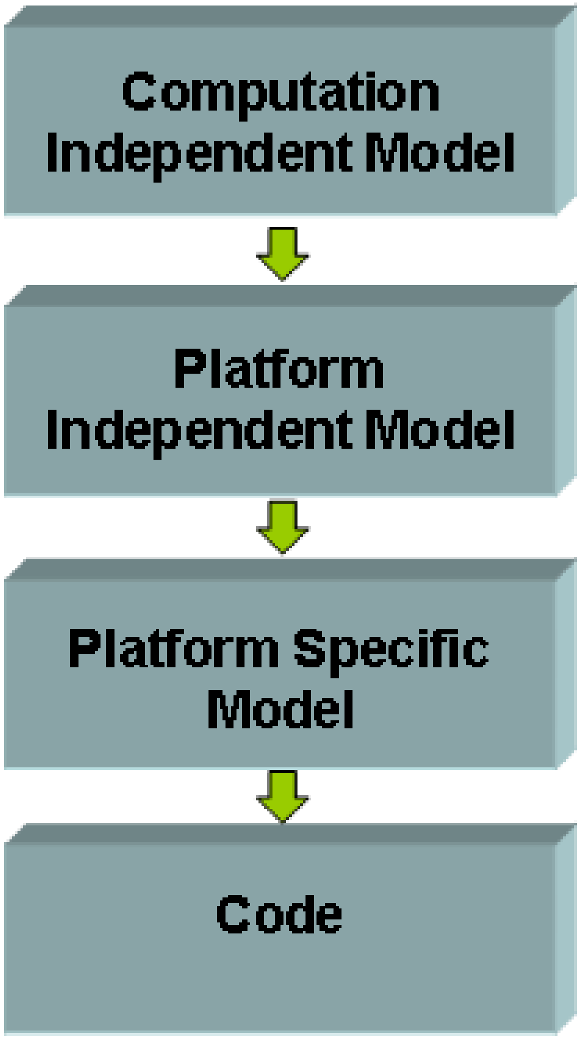
- **To specify SOSs**
  - structure, behaviour, ...
  - separate concepts at different conceptual levels
  - communicate with stakeholders
- **To understand the SOS**
  - if existing (legacy applications)
- **To validate SOSs**
  - detect errors and omissions in design ASAP
  - prototype the system (*execution* of the model)
  - formal analysis of system properties
- **To drive implementation**
  - code skeleton and templates
  - complete programs (if possible)



Excursion:
Model-driven development

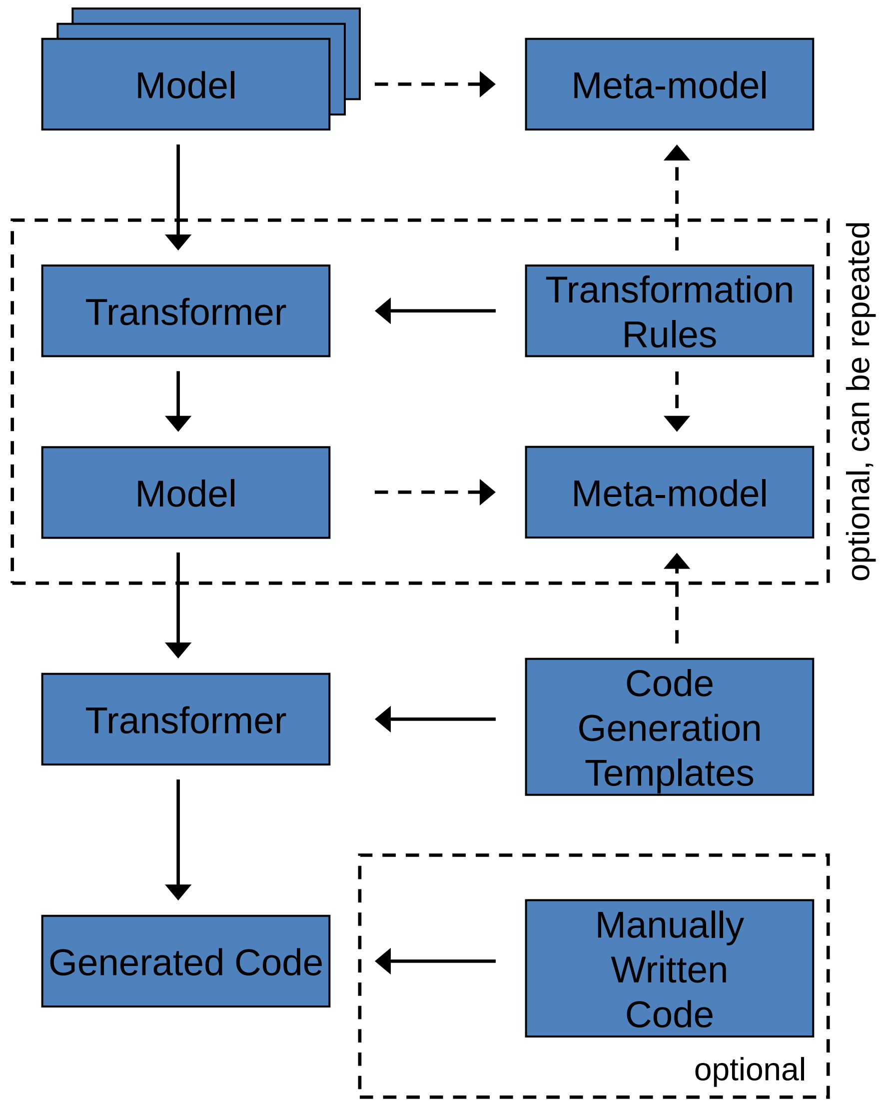
„The Architecture of Choice for a Changing World“ [OMG, 2001]

- Model Driven Architecture®
  - *Specify* a system independently of its platform
  - *Specify* and choose a platform for the system
  - *Transform* the system specifications into a platform dependent system

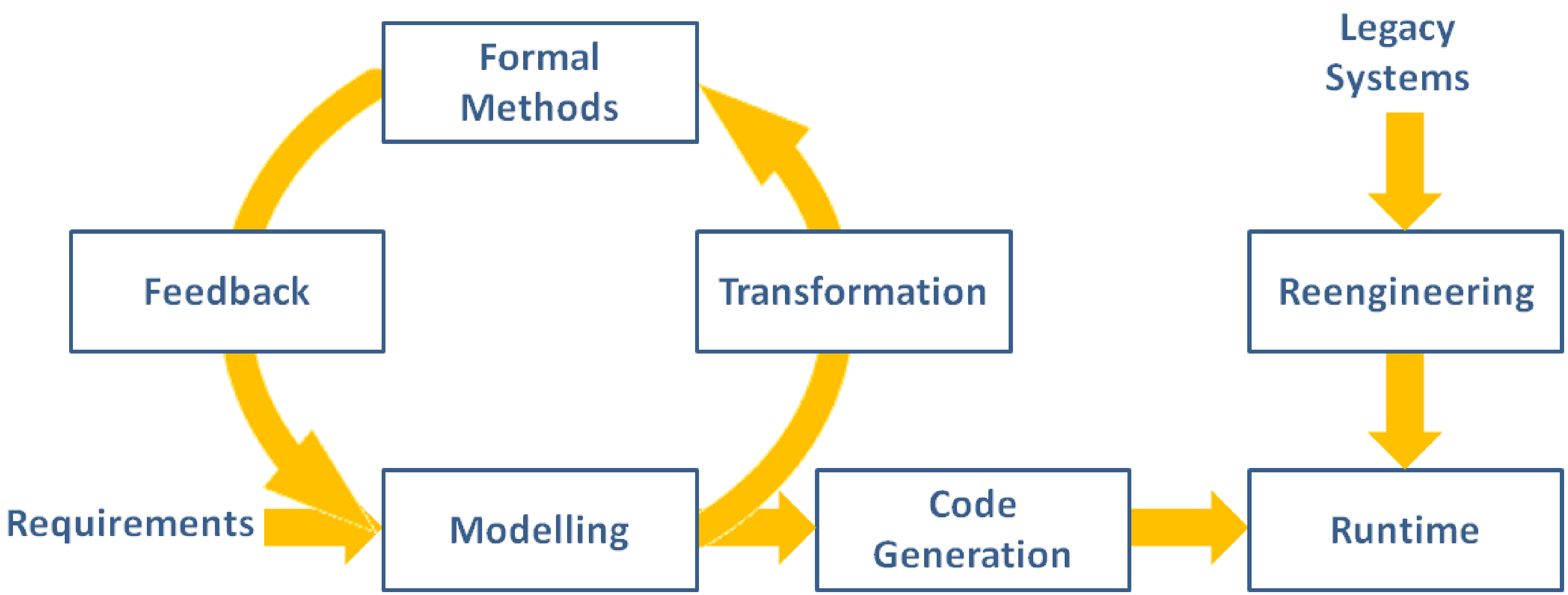


Excursion: MDA Approach

- Choose a domain-specific language for each layer
- Use meta-models to describe languages
- Use model transformations to convert models
  - Model-to-model transformations
    - Transformations may be between different languages. In particular, between different languages defined by MOF
  - Model-to-text transformations
    - Special kind of model to model transformations

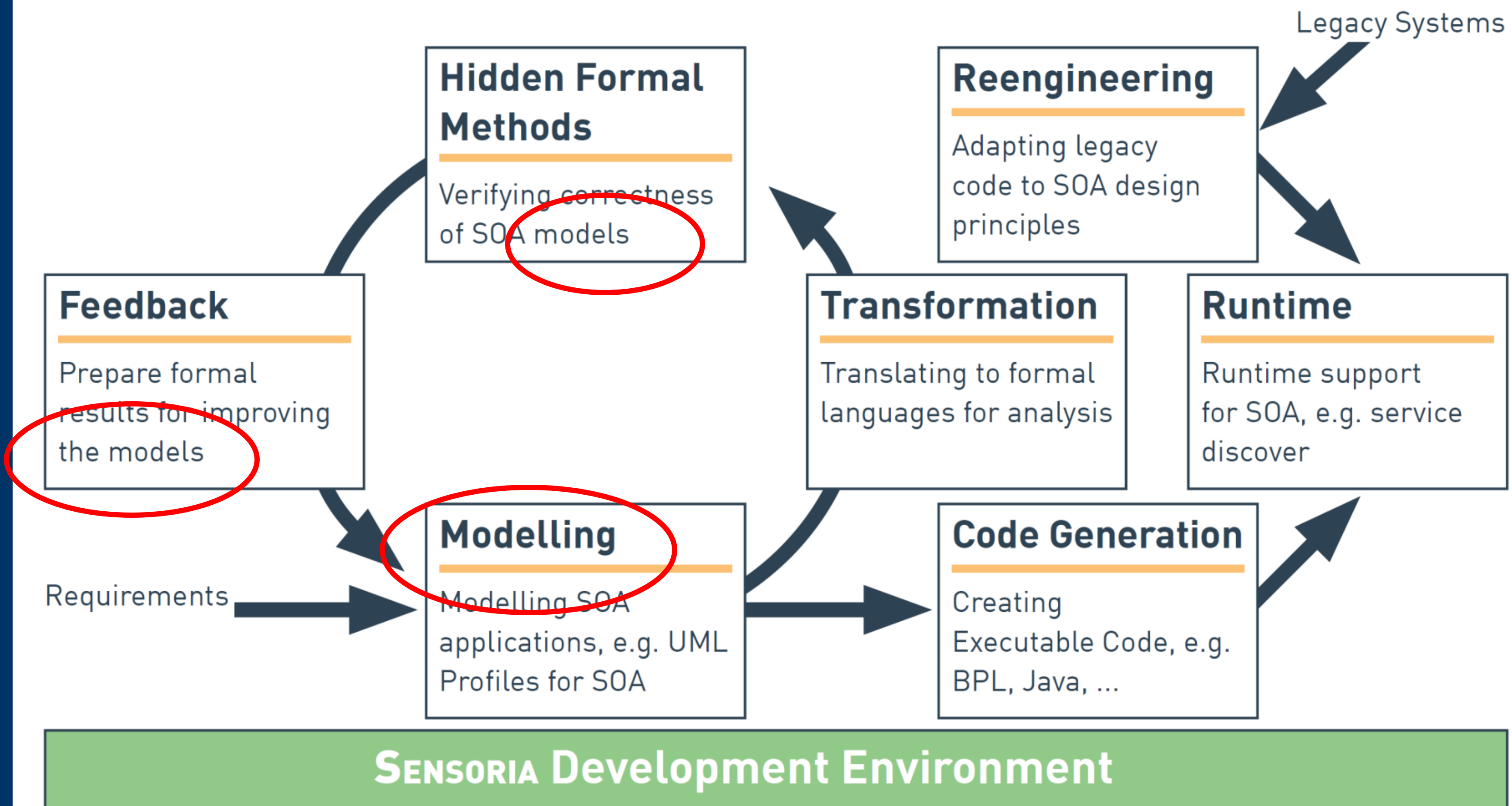


SENSORIA Model-driven development



SENSORIA Model-driven development

Details

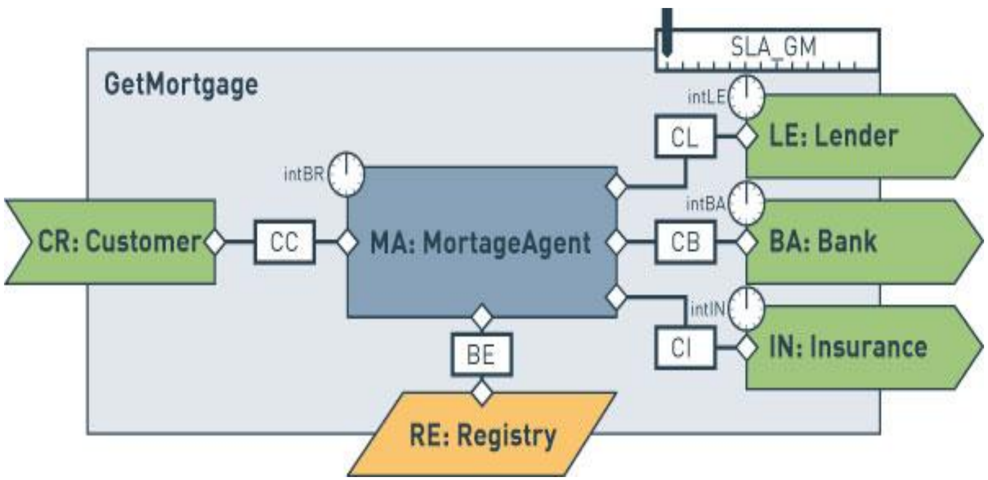


Modelling languages

- Objective is to have a domain specific graphical representation and clear semantics for service-oriented concepts
  - **Option 1:** Definition of a proprietary language, like SENSORIA Reference Modelling Language (SRML)
    - high cost: requires the definition of all required domain specific concepts and proprietary tools
  - **Option 2:** Use of a standard, like Unified Modeling Language (UML™), Business Process Modeling Notation (BPMN™)
    - diagrams are more difficult to read and/or not integrated into UML
  - **Option 3:** Define a UML2 profile
    - using the extension mechanism that allows to customize the UML for specific domains and platforms
    - defining stereotypes, stereotype attributes (tagged values) and constraints to restrict and extend the scope of UML
    - UML CASE tools can be used

Option 1:
SENSORIA Reference Modelling Language (SRML)

- Modelling language with a formal semantics
- Offers descriptions of business logic based on conversational interactions
- Inspired by SCA (standards proposed by IBM, BEA, Oracle, SAP, Siebel,…)
- Proprietary language needs proprietary CASE tool




```
INTERACTIONS
r&s getProposal
    idData:usrdata,
    income: moneyvalue,
    preferences:prefdata
    proposal:mortgageproposal,
    cost:moneyvalue
```

www.sensoria-ist.eu
Teaching material, tutorial, June 2009

Option 3: UML2 profile

- **Main Aim:** *to have a powerful yet readable graphical modelling language for SOAs – based on UML*
  - *"minimalist" extension*
    - use UML constructs wherever possible
    - use other extensions if available
    - only add new model elements where needed
  - *reducing efforts of building SOA models*
    - covering domain specific aspects, such as
      - service contracts
      - long running transactions and compensation
      - loose coupling of services

➲ UML4SOA

- **Secondary Aim**: *to employ transformers from such models to common implementation languages* (BPEL, Java...)

➲ MDD4SOA

UML extensions for SOA modelling

- SoaML profile (OMG open source specification)
  - Service-oriented architecture Modeling Language
  - for structural aspects of services
- UML4SOA profile (developed within the scope of the project)
  - for behavioural aspects, e.g. orchestration
  - for non-functional aspects
  - for reconfiguration
  - for policies
  - for requirements
- MARTE profile (OMG standardization process beta2 version)
  - for performance analysis

UML4SOA, SoaML, MARTE

- Defined as UML profiles
  - provide a set of elements for modelling SOAs
  - use UML extension mechanisms (stereotypes)
  - no changes to UML (exception SoaML propose one change)
- Use of the profiles
  - to build models at different levels of abstraction
  - in combination with UML model elements
  - is not a prescriptive approach

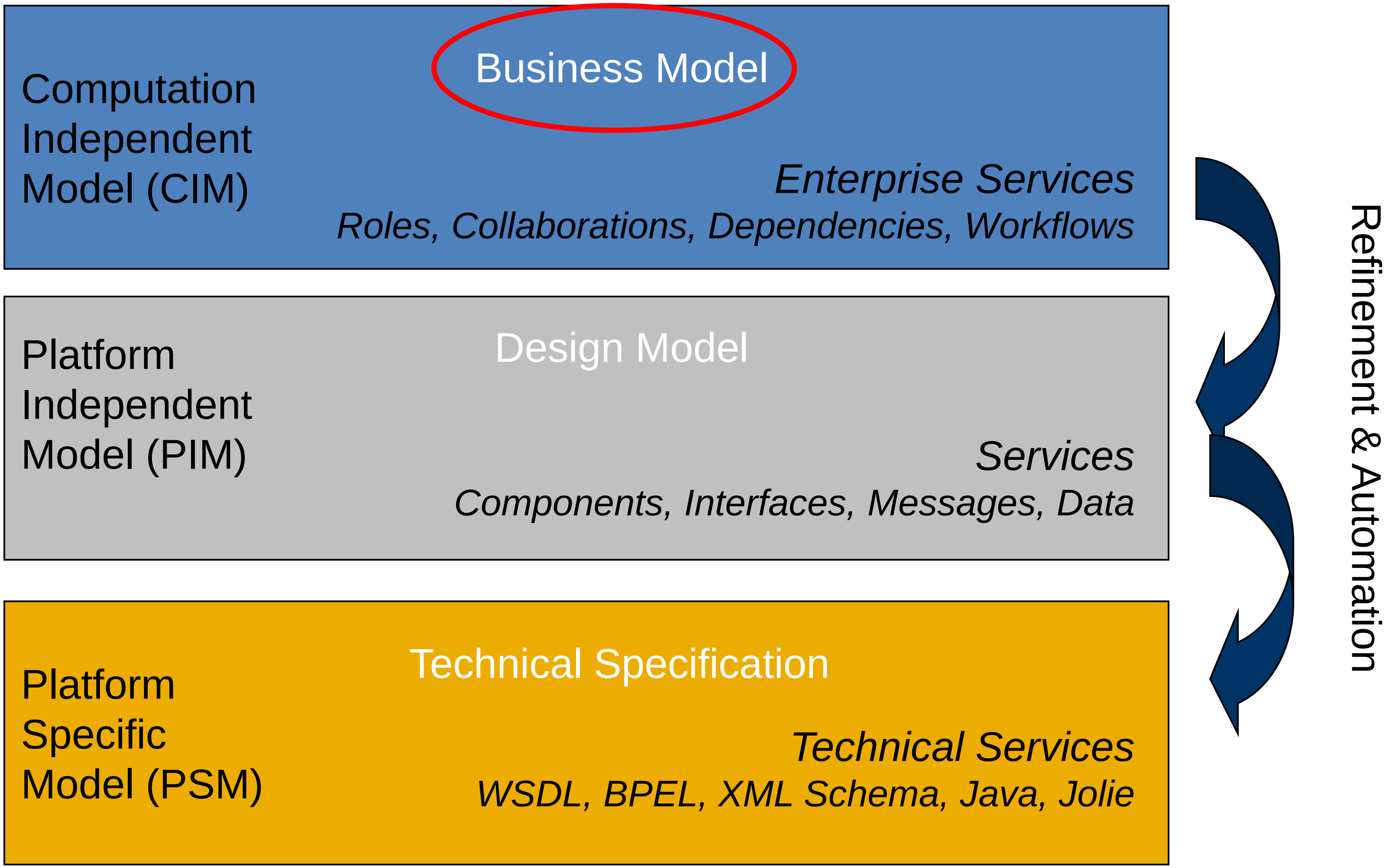
SoaML

- Answer to Request of Proposal of the OMG
  - for a *UML Profile and Metamodel for Services* (UPMS), Sept. 2006
- Submission and supporters
  - SINTEF, Norway (co-ordination), European Software Institute (ESI)
  - Capgemini, Fujitsu, Hewlett-Packard, IBM, Telelogic AB, Thales Group, France Telecom R&D, etc
  - University of Insbruck, University of Augsburg, University of Athens
  - SHAPE project (FP7) is the main contributor
- Meetings SoaML and UML4SOA groups
  - EDOC 2008, Munich, Sept. 2008
- SoaML standardized, version 1.0, March 2012

MARTE profile

- Defined for modelling of real-time and embedded systems
- Concerns also model-based analysis, i.e. provides facilities to annotate models with information required to perform specific model analysis
- Focuses on performance and schedulability analysis

SOA models in the MDA context



Source: Data Access Technologies, Inc

SOA modelling by example

- Finance Case Study: Credit Portal Scenario
  - Stakeholders (parties) of the service-based scenario are customers, clerks and supervisors.
  - Login is required, if a customer wants to request a credit by using the credit portal.
  - The credit request process requires from the customer credit data, security data and balance data
  - Based on the uploaded information the system calculates a rating that is used for an automatic decision, a clerk or supervisor decision.
  - In case of a positive decision the process informs the customer and waits for his decision.
  - Once the credit offer is accepted, the process stores the credit offer in an agreement system and the process is finalised.
  - In case of a negative decision the customer is informed about this decision and the process ends, too.

Constructing the business model

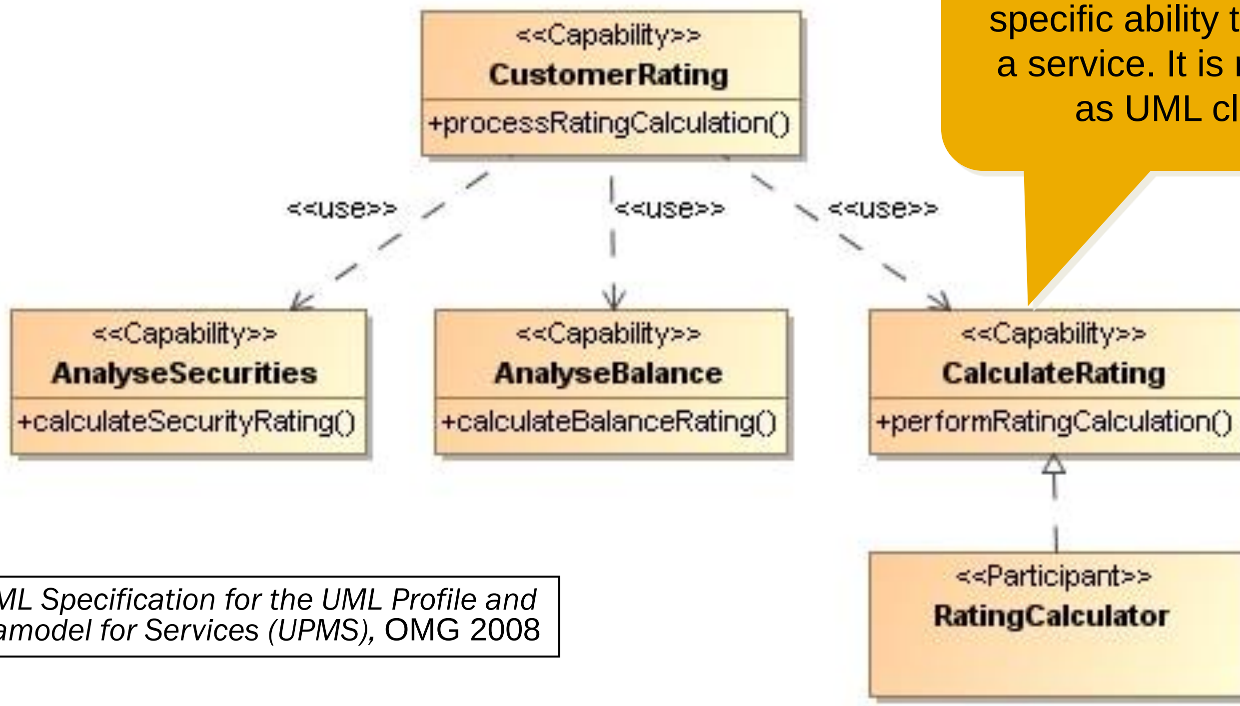
1. Specify the needed service capabilities
   - identify the needed services and
   - organize them into catalogues
2. Identify the parties involved
   - identify the provider and consumers of services
3. Model the service contracts
   - specify the agreement between providers and consumers of a service
4. Build service architecture
   - describe how participants work together for a purpose by providing and using services expressed as service contracts

Specifying service capabilities

SoaML

- Capabilities are used
  - to identify needed services
  - to organize them into catalogues or network of capabilities
  - prior to allocating those services to particular service providers and requesters

A ***capability*** is the specific ability to provide a service. It is modelled as UML class.



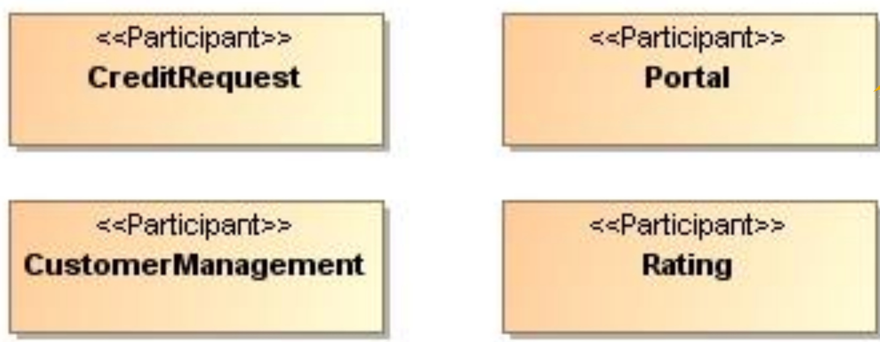
SoaML Specification for the UML Profile and Metamodel for Services (UPMS), OMG 2008

Identifying parties involved in SOAs

SoaML

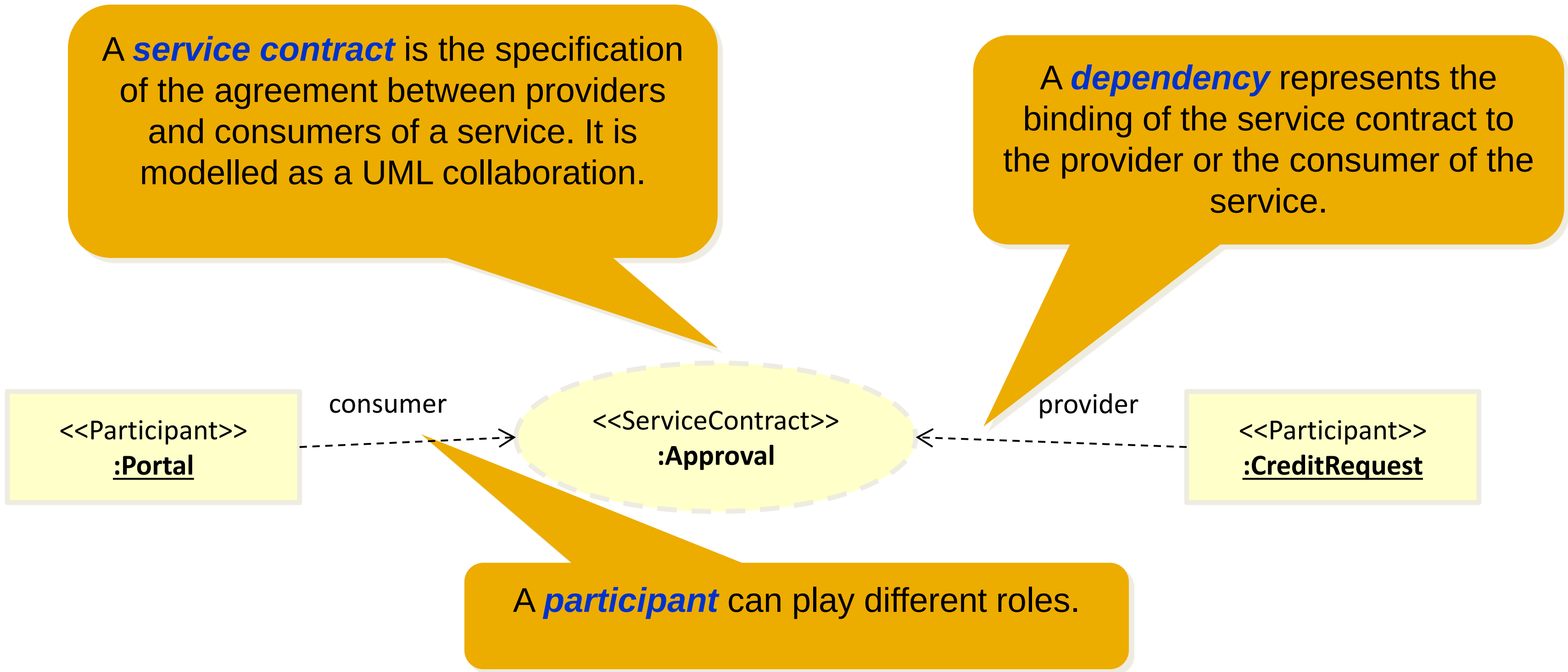
- Provider and consumers of services are represented as participants
  - in the business domain: person, organization or system
  - in the systems domain: system, application or component
- Participant can play the role of
  - providers in some interactions
  - consumers in others

A ***participant*** represents some party that provides and/or consumes services. It is modelled as UML class.



Modelling service contracts

SoaML

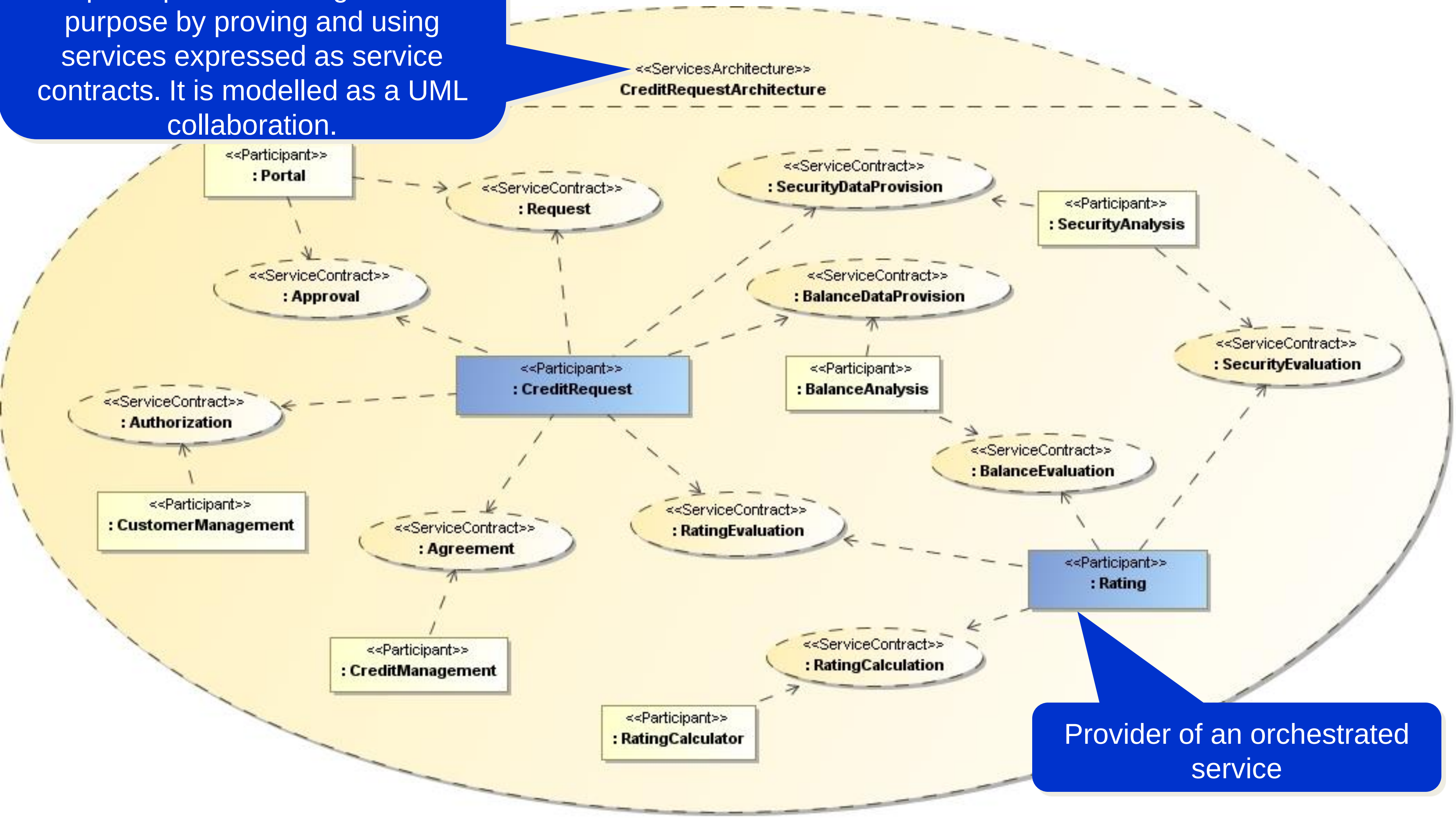


- A service contract specifies the service without regards for realization or implementation.
- A UML2 collaboration defines a set of cooperating entities to be played by instances (its roles), as well as a set of connectors that define communication paths between the participating instances.

Representing service architecture

SoaML

A ***service architecture*** describes how participants work together for a purpose by proving and using services expressed as service contracts. It is modelled as a UML collaboration.



Provider of an orchestrated service

SOA models in the MDA context

SoaML

Computation Independent Model (CIM)

Business Model

Enterprise Services

Roles, Collaborations, Dependencies, Workflows

Platform Independent Model (PIM)

Design Model

Services

Componentes, Interfaces, Messages, Data

Platform Specific Model (PSM)

Technical Specification

Technical Services

WSDL, BPEL, XML Schema, Java, Jolie

Refinement & Automation

Source: Data Access Technologies, Inc

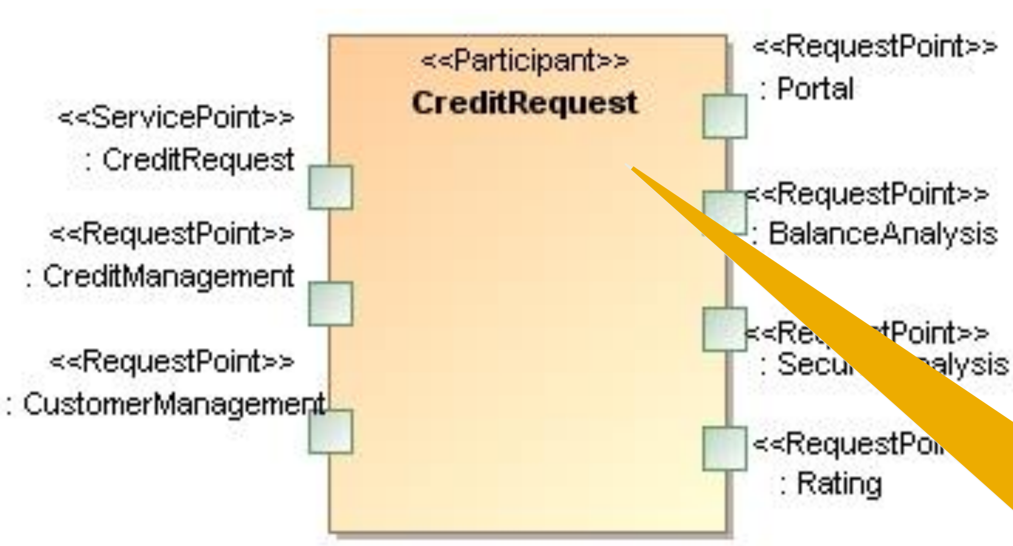
Constructing the design model

- **Refine the specifications of participants with ports**
  - for provided and consumed services
- **Model the service interfaces**
  - Classify ports into service points (for providing services) and request points (for consuming services)
  - Define the service interfaces
    - structurally by inheritance from UML Interfaces
    - behaviorally by protocol state machines
- **Specify the orchestration of the services**
  - i.e. combine existing services to build the required new services
  - by UML4SOA activity diagrams
    - Including partner services, message passing among requester and provider, and long-running transactions
- **Define the quality of service** (service level agreements)
  - by specifying the required non-functional properties

Refining specification of participants with ports

SoaML

- Add ports for provided and consumed services
- A port has as type a service interface or an interface



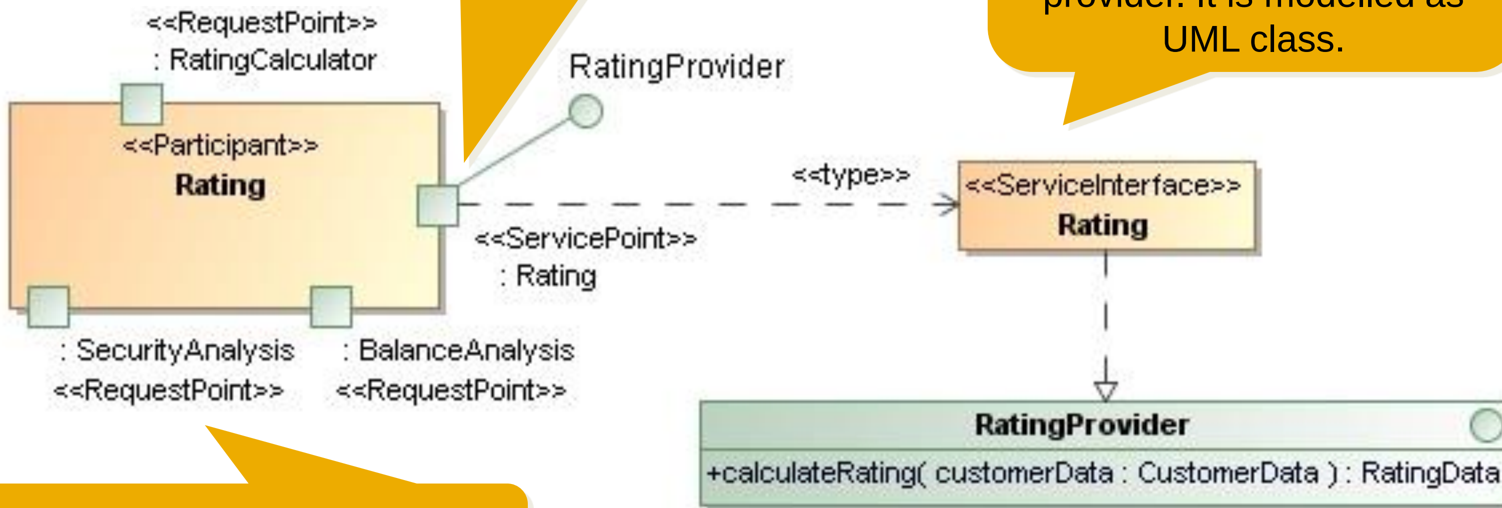
A full specification of a ***participant*** includes ***ports*** for every service contract in which the participant participates within the service architecture. Two types of ports: ***service point*** and ***request point***

Modelling service interfaces

SoaML

A ***service point*** is a port for providing a service.

A ***service interface*** allows for connection between the service consumer and provider. It is modelled as UML class.



A ***request point*** is a port for requesting (consuming) a service

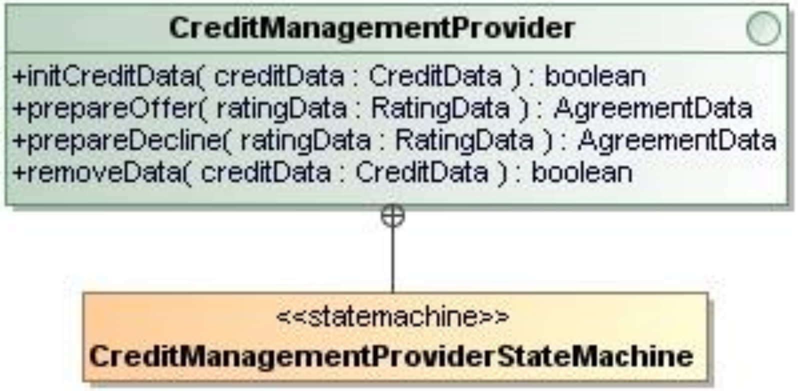
A ***UML interface*** is used to represent the required and provided interfaces of the ports.

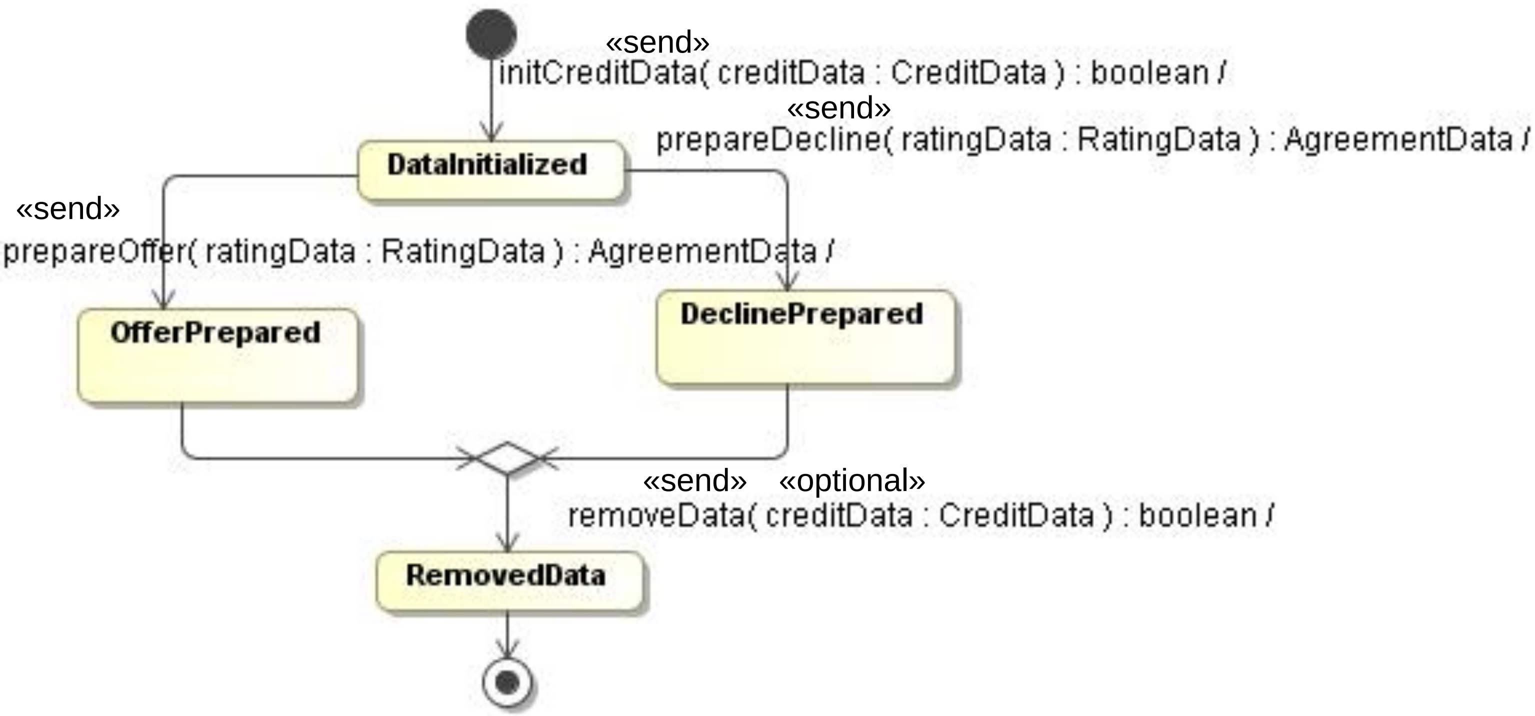
- A service interface
  - “provides” provider interfaces (represented as realisation)
  - “requires” consumer interfaces (represented as a «use» dependency)

Interface behaviour

UML

- UML4SOA
  - proposes **protocol state machines**
- Remark
  - SoaML proposes activity diagrams or sequence diagrams

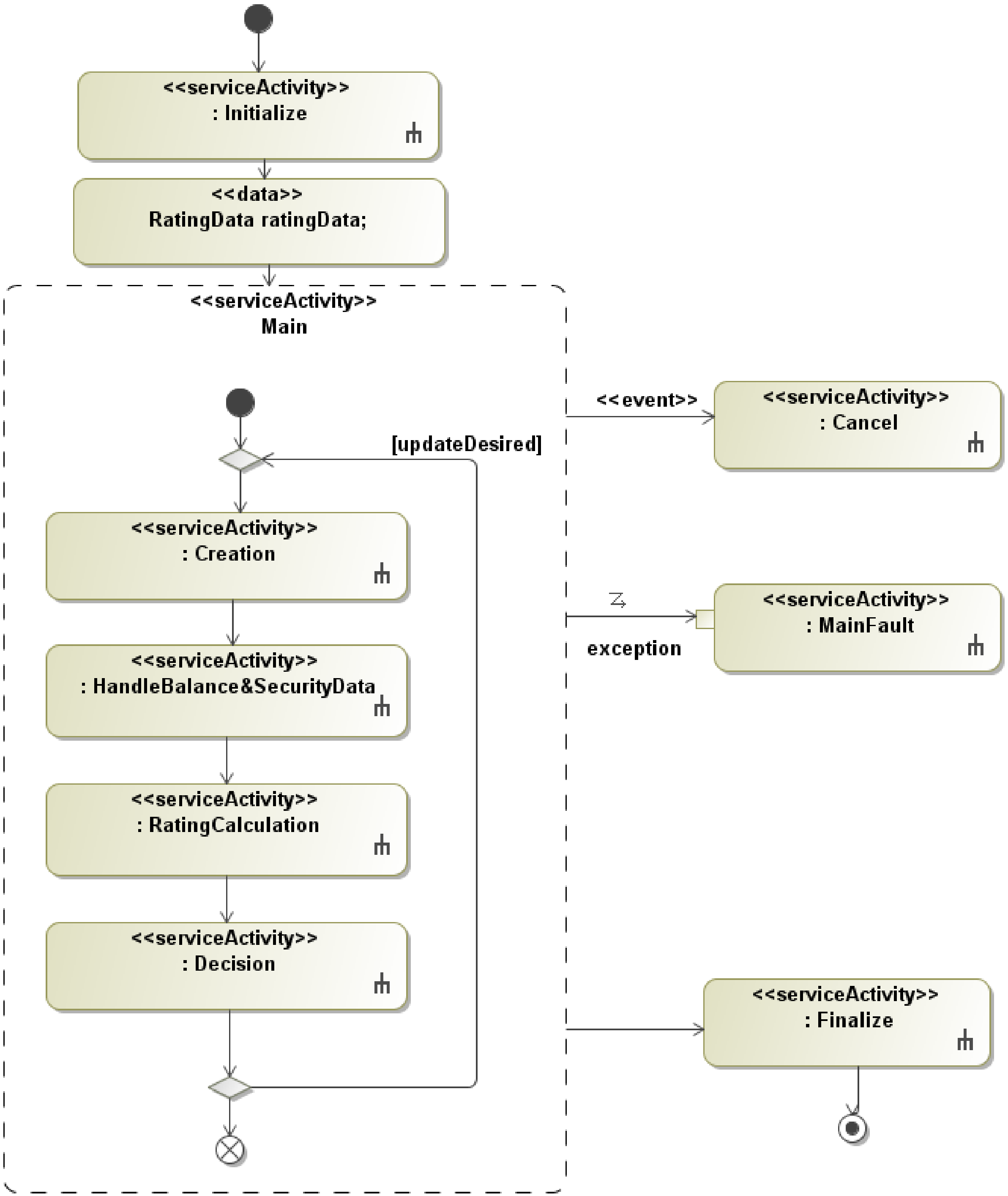




Orchestration of

- Service orchestrati
the process of
combining existing
services to form a
service to be used
any other service.

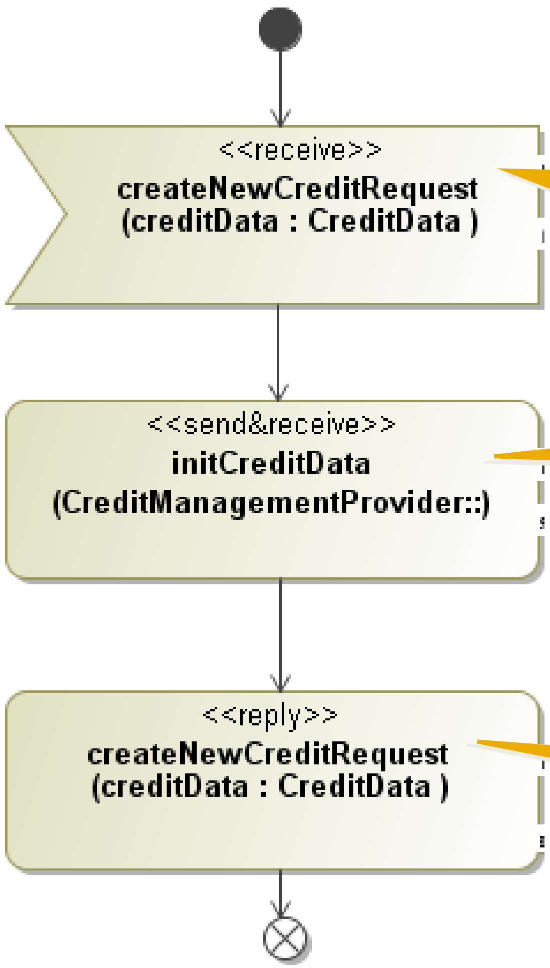
- Key distinguishing
concepts
  - partner services
  - message passing
among requester
provider
  - long-running
transactions
  - compensation



Message passing

UML4SOA

Synchronous and asynchronous service invocation



Service interaction ***send*** sends a message. Does not block.

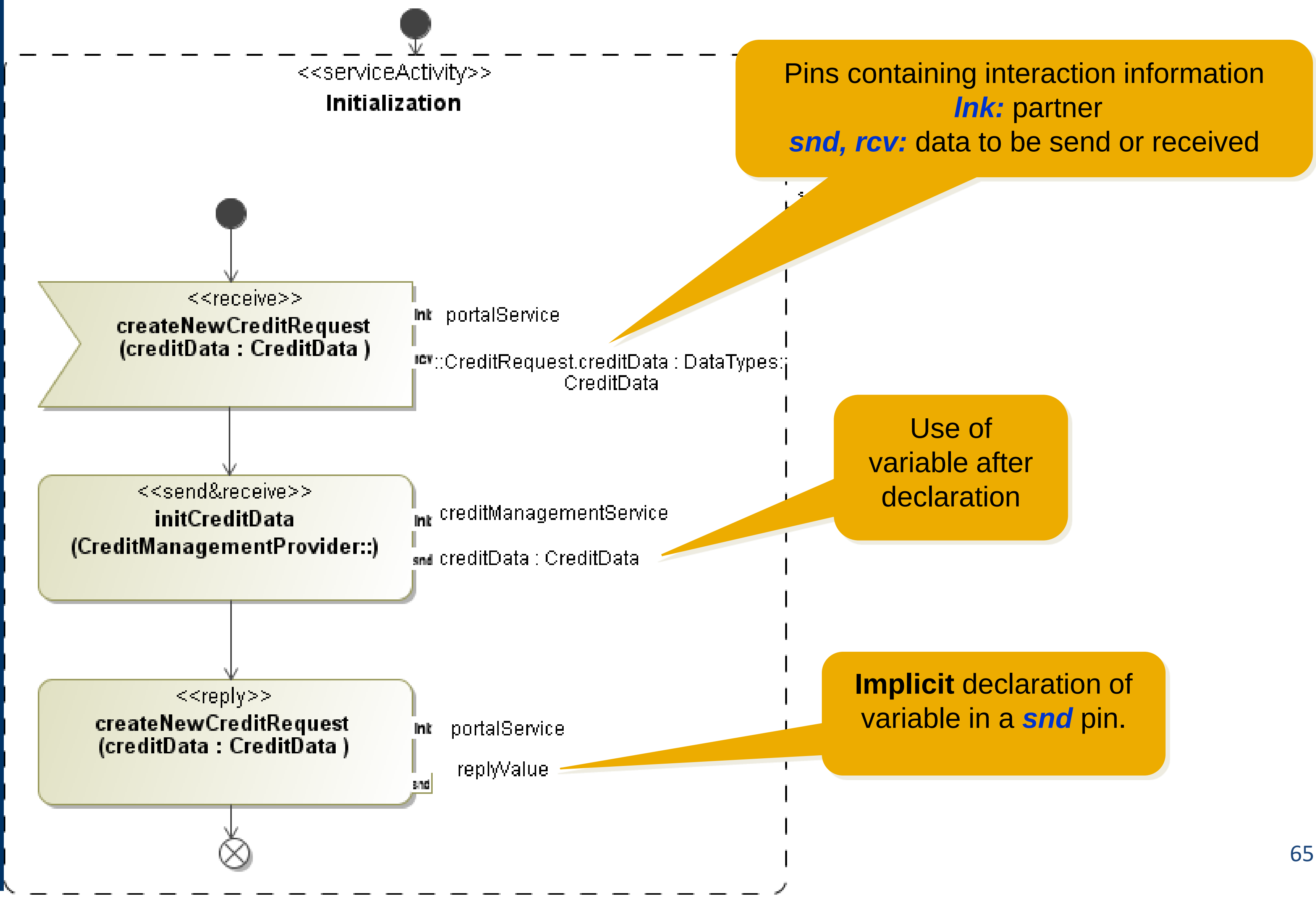
Service interaction ***receive*** blocks until message is received.

Service interactions ***send&receive, receive&send*** denotes a sequential order of these actions.

Reply is used for the reception of a message decoupled of the sending process

Detailing service invocation
Partner services and data handling

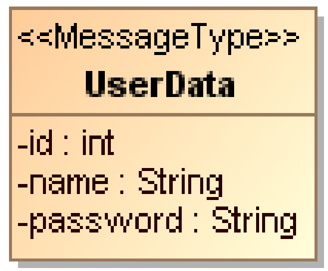
UML4SOA



Data handling

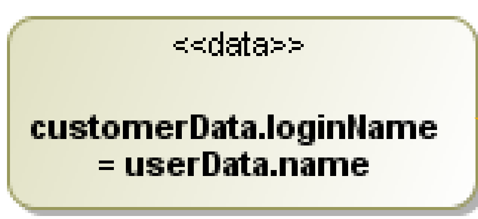
SoaML/UML4SOA

- Declaration of structured types
  - extends metaclass data type and class



A ***message type*** is used to specify information exchanged between service consumers and providers (message passing).

- Use in behavioural diagrams
  - support for typed, scoped variables in the orchestration
  - data handling support

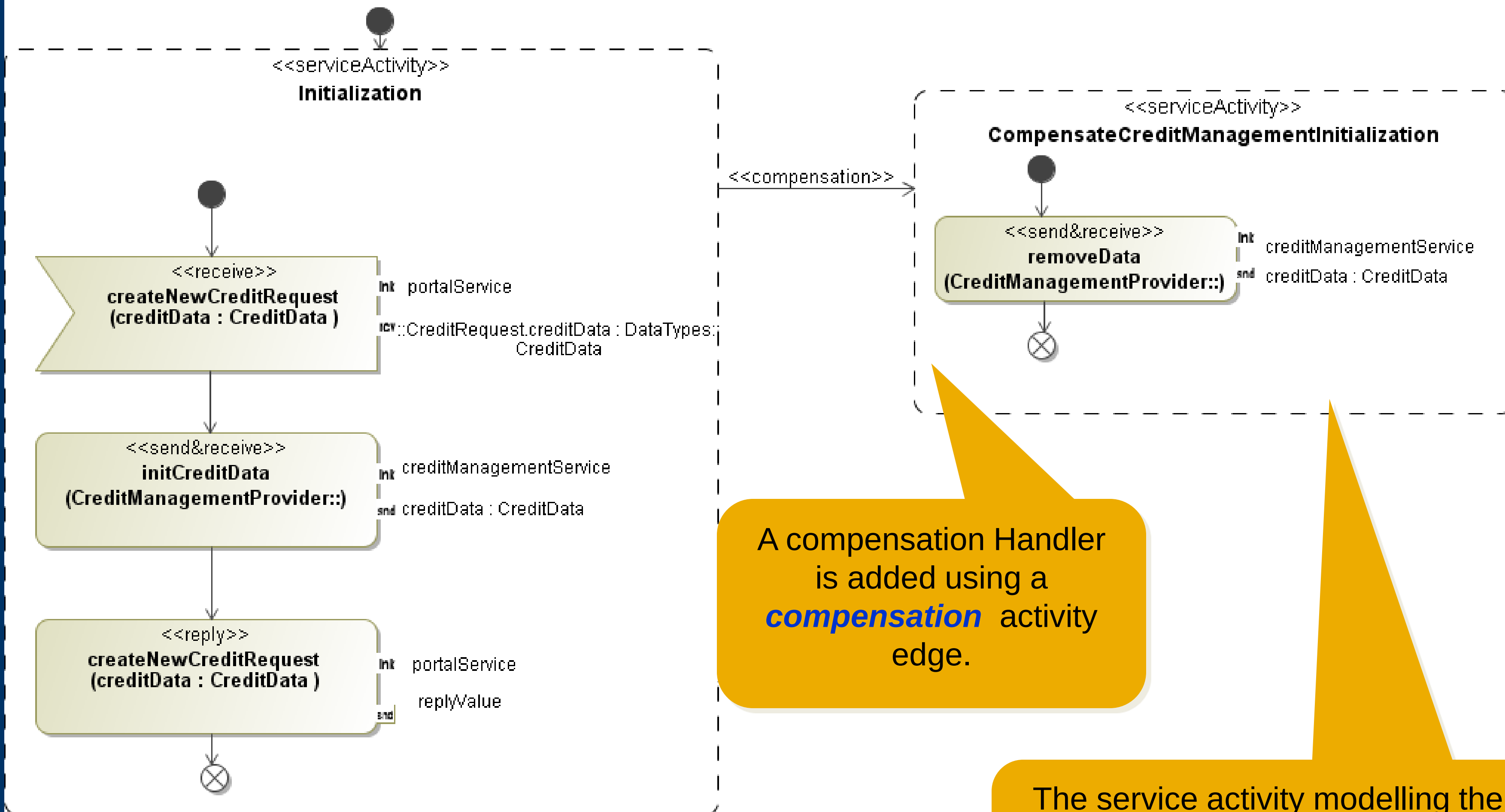


A ***data action*** can be used to **explicitly** declare the type of a variable or for **manipulation** of data (copy, calculation, etc).

Long running transactions

UML4SOA

- Require compensation mechanisms, e.g. compensation handlers

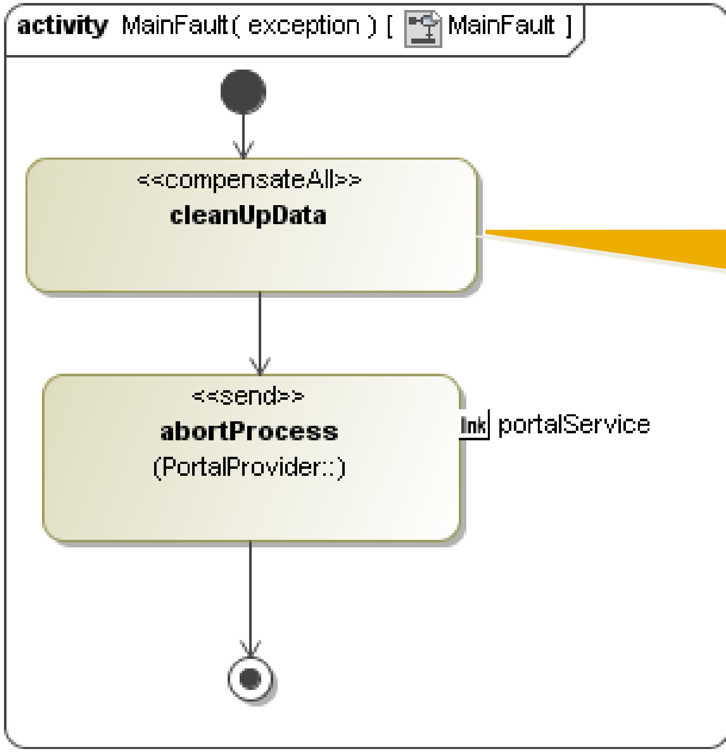


A compensation Handler is added using a ***compensation*** activity edge.

The service activity modelling the compensation handler will be triggered by a ***compensate*** or ***compensateAll.***

Compensation

UML4SOA



A ***compensateAll*** triggers all active compensation handlers in the reverse order.

SOA model elements and diagram types

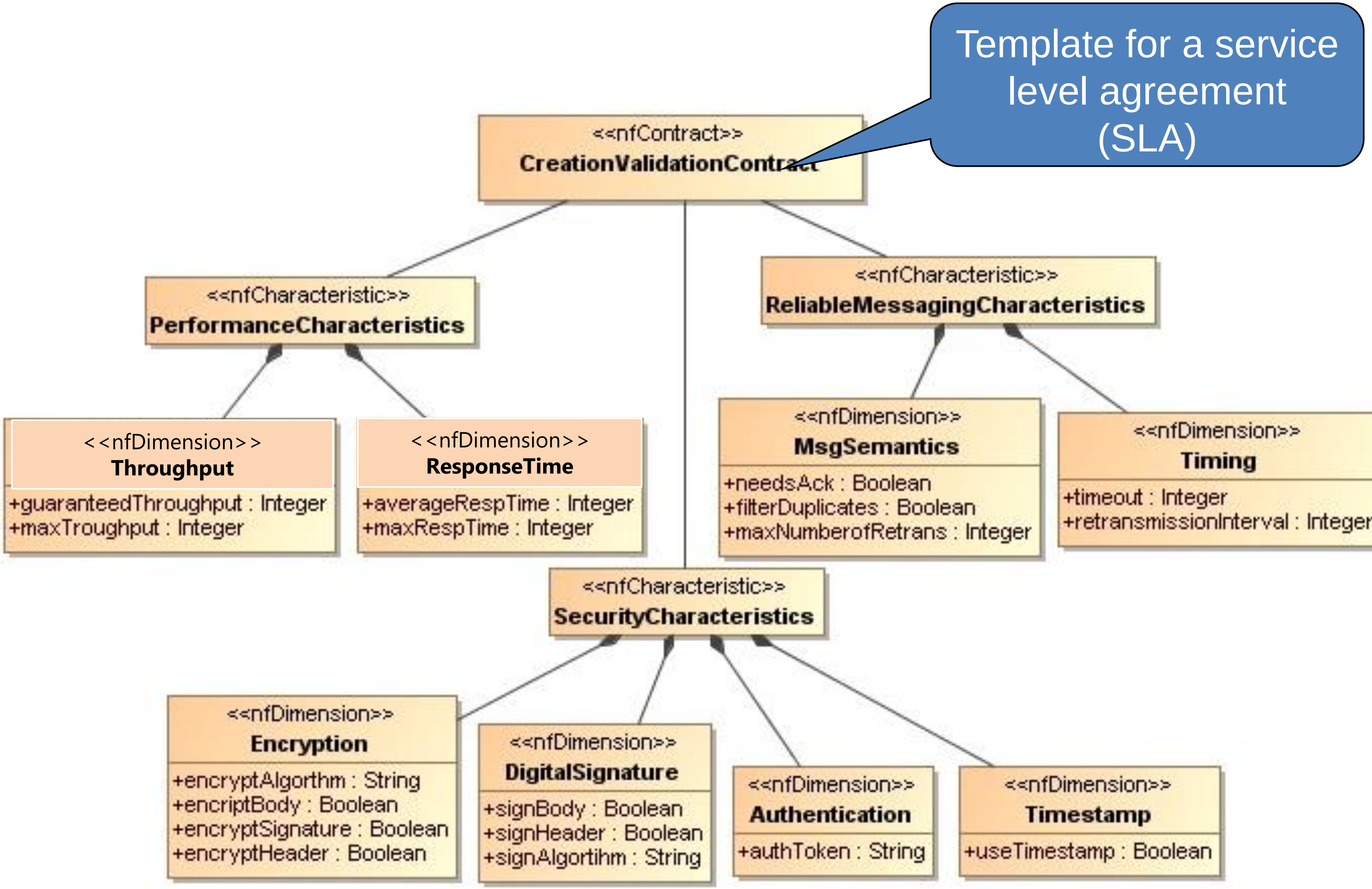
	Business model	*Design model*
Structural aspects	capabilities participants service contract service architecture participant architecture	service point request point service interface message type
Behavioural aspects	scope	send, receive, send&receive reply, raise lnk, snd, rcv compensate, compensateAll compensation, exception, event data
Diagram type	class diagram composite structure diagram activity diagram	class diagram composite structure diagram activity diagram sequence diagram state machine

+ use of plain UML, e.g. SOA's protocols

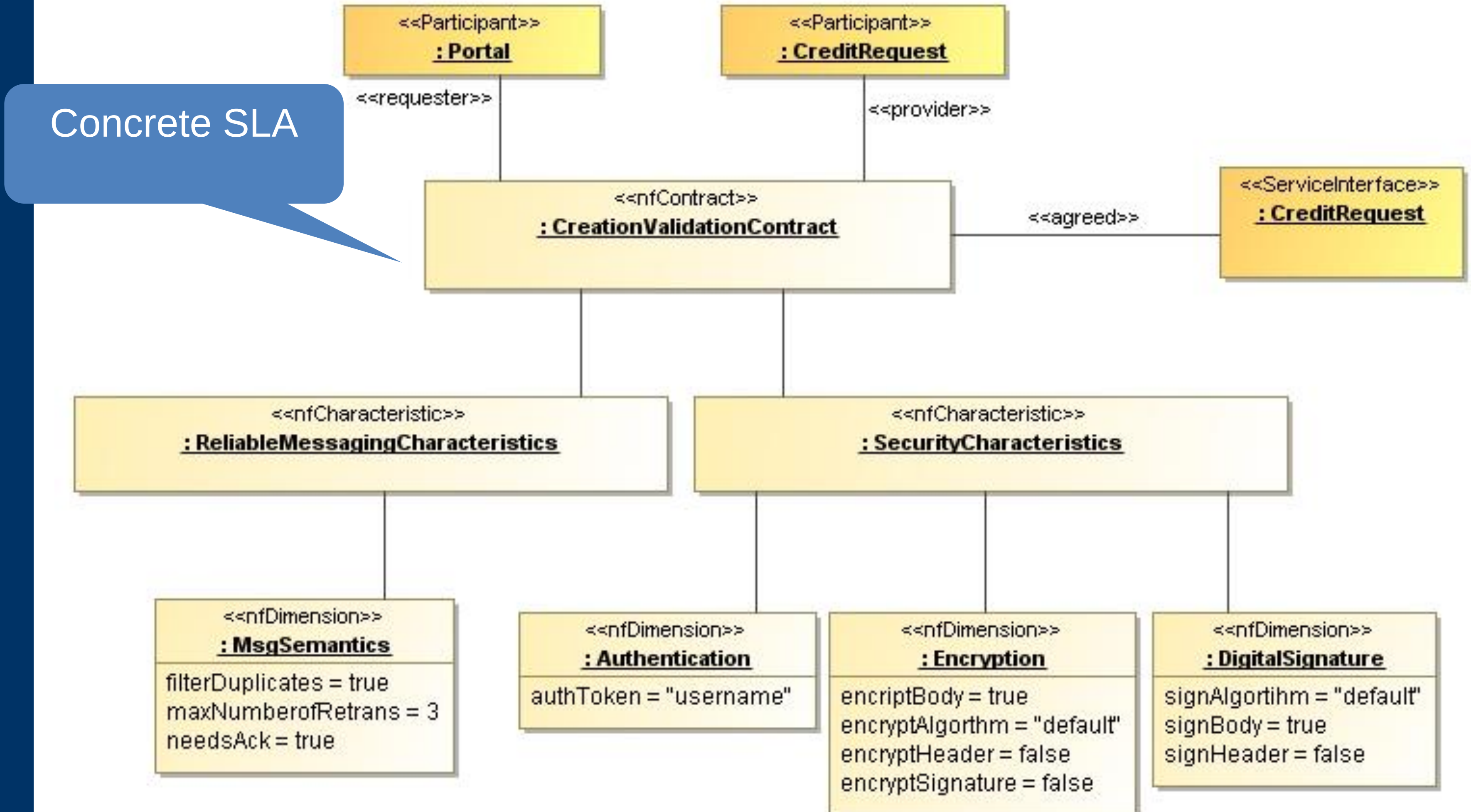
Quality of services

- Defined by non-functional properties (NFP)
- Example: Credit Portal Scenario
  - The *Portal* and the *CreditRequest* should communicate via a secure and reliable connection
  - All requests sent to the *CreditRequest* should be acknowledged
  - As the credit request handles confidential data, all requests should be encrypted in order to protect the privacy of the customers
  - Messages sent by the *CreditRequest* must be clearly accountable, i.e. non-repudiation of messages must be guaranteed

Modelling approach for NFP of services



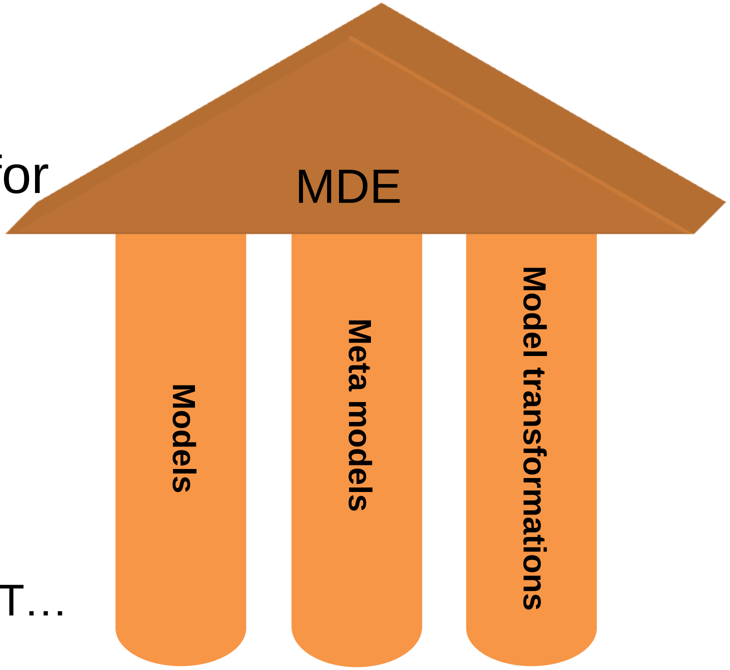
Modelling a concrete configuration



Coming back to MDE

- MDE approaches
  - are based on the constructions of models
  - propose transformation of models
  - implement model transformations based on the metamodel of the modelling language

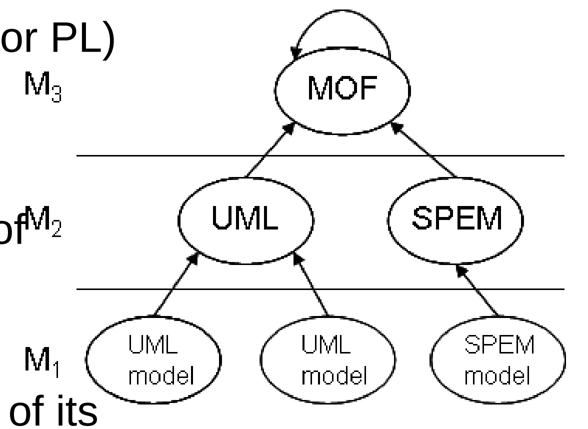
- MDE approaches require languages for
  - specification of models
    - UML, BPMN, …
  - description of metamodels
    - UML, MOF, OCL, …
  - definition of model transformations
    - Java, graph transformations, ATL, QVT…



Metamodels

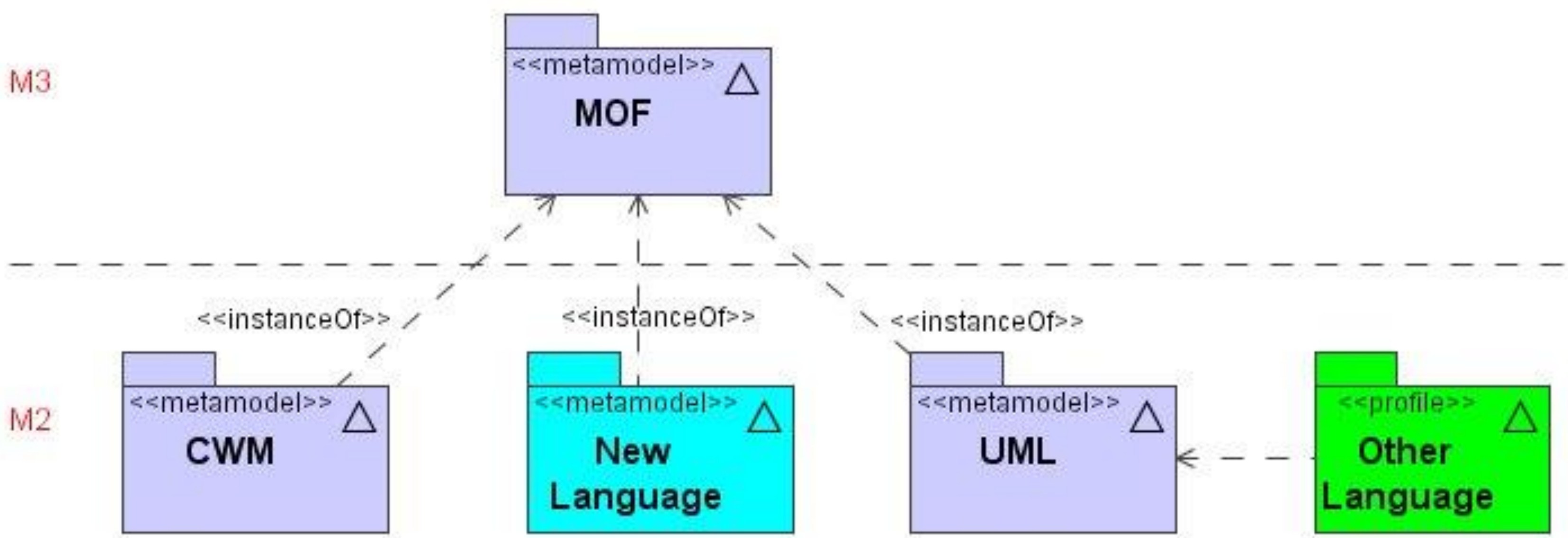
- A *metamodel* of a domain is a description of the concepts of this domain and their relationships
  - Metamodels formalize the syntax of (Software Engineering) models
  - Metamodels are the equivalent of (context free) grammars of programming languages

- Example UML: a three layer structure
  - (M3) Meta-metamodel: *Meta-Object Facility* (MOF)
    - formalizes the syntax of UML (similar to BNF for PL)
    - is some kind of "top level ontology"
  - (M2) Metamodel
    - Defines *structure* and *constraints* for a family of models.
  - (M1) Model
    - Each of the models is defined in the language of its *unique meta-model*.



Language definition mechanisms

- Options for defining a new modelling languages
  - New MOF-based modelling language
  - UML extension (profile)



UML Profile

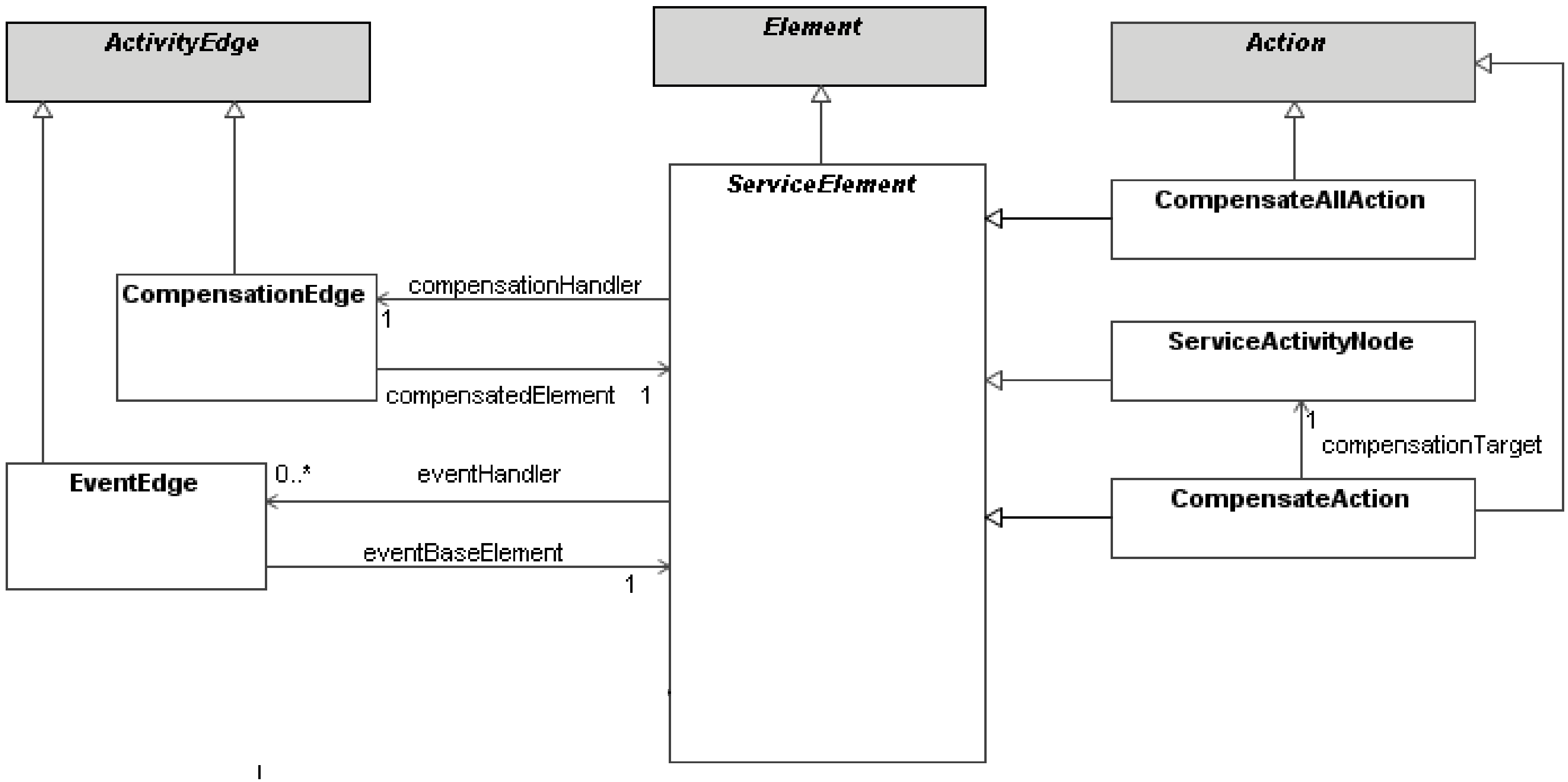
- Extension of the UML for domain specific model element
  - providing a different notation
  - enriching model elements with additional semantics (e.g. request point)
  - representation of domain specific patterns (e.g. compensation)
  - annotations (marks) facilitating model transformations in a model-driven approach (e.g. lnk)
- Use of extension mechanisms of the UML
  - stereotypes
  - tagged values
  - constraints
- Risks
  - too many stereotypes
  - selection of inadequate UML metaclass
  - decorative and redefined stereotypes (→)

Creating a UML profile

- Specification of a metamodel for the specific domain
  1. identification of the domain specific concepts and their relationships
  2. construction of a model capturing concepts and relationships (metamodel)
  3. UML elements for this concepts? (minimalist extension)
- Specification of the profile
  1. creation of stereotypes for identified elements
  2. identification of appropriate UML metaclasses
  3. stereotypes and metamodel elements related by an “extension” (multiple metaclasses)
  4. define semantics of new elements

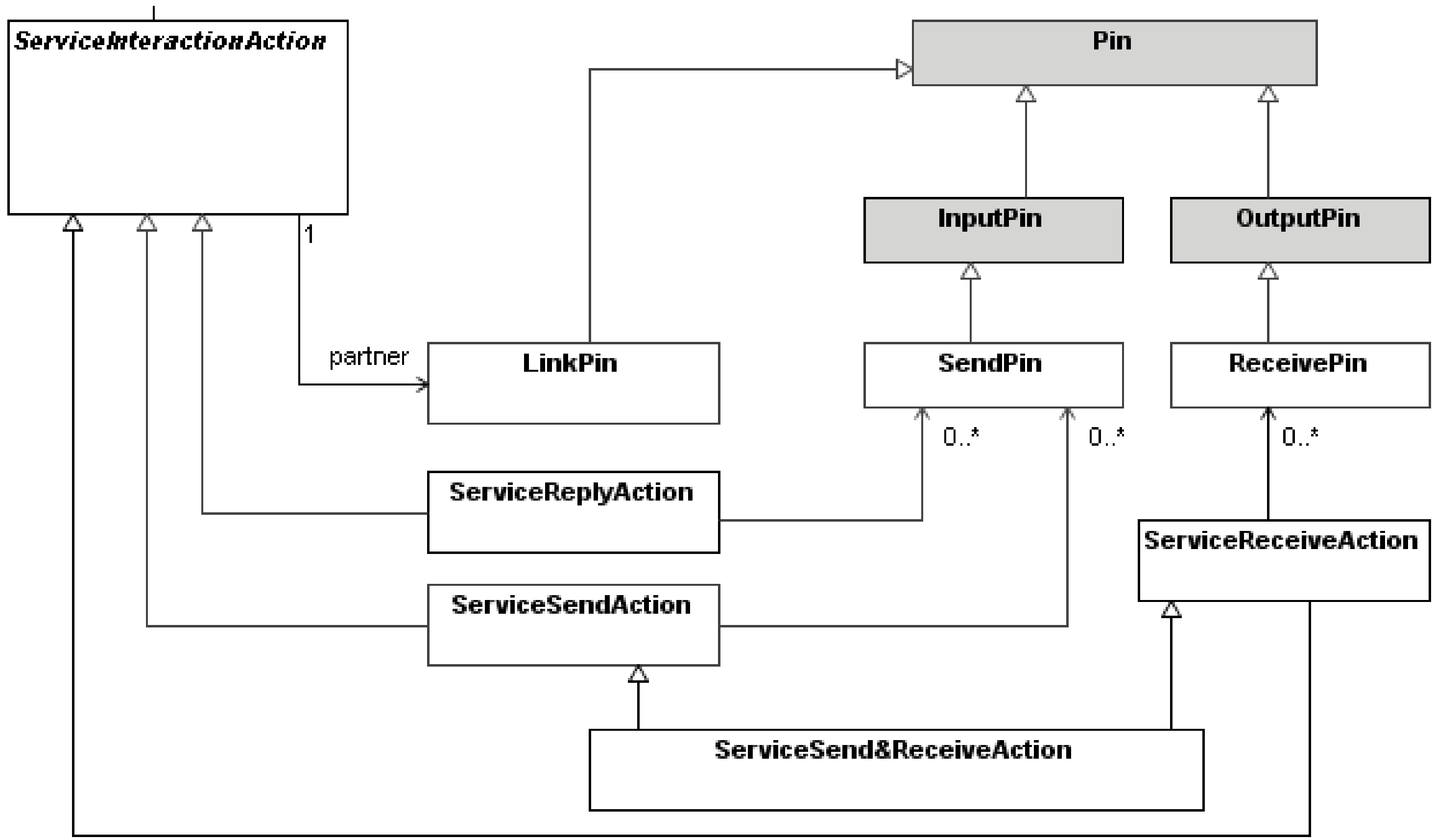
UML4SOA metamodel: Orchestration

Conservative extension of the UML

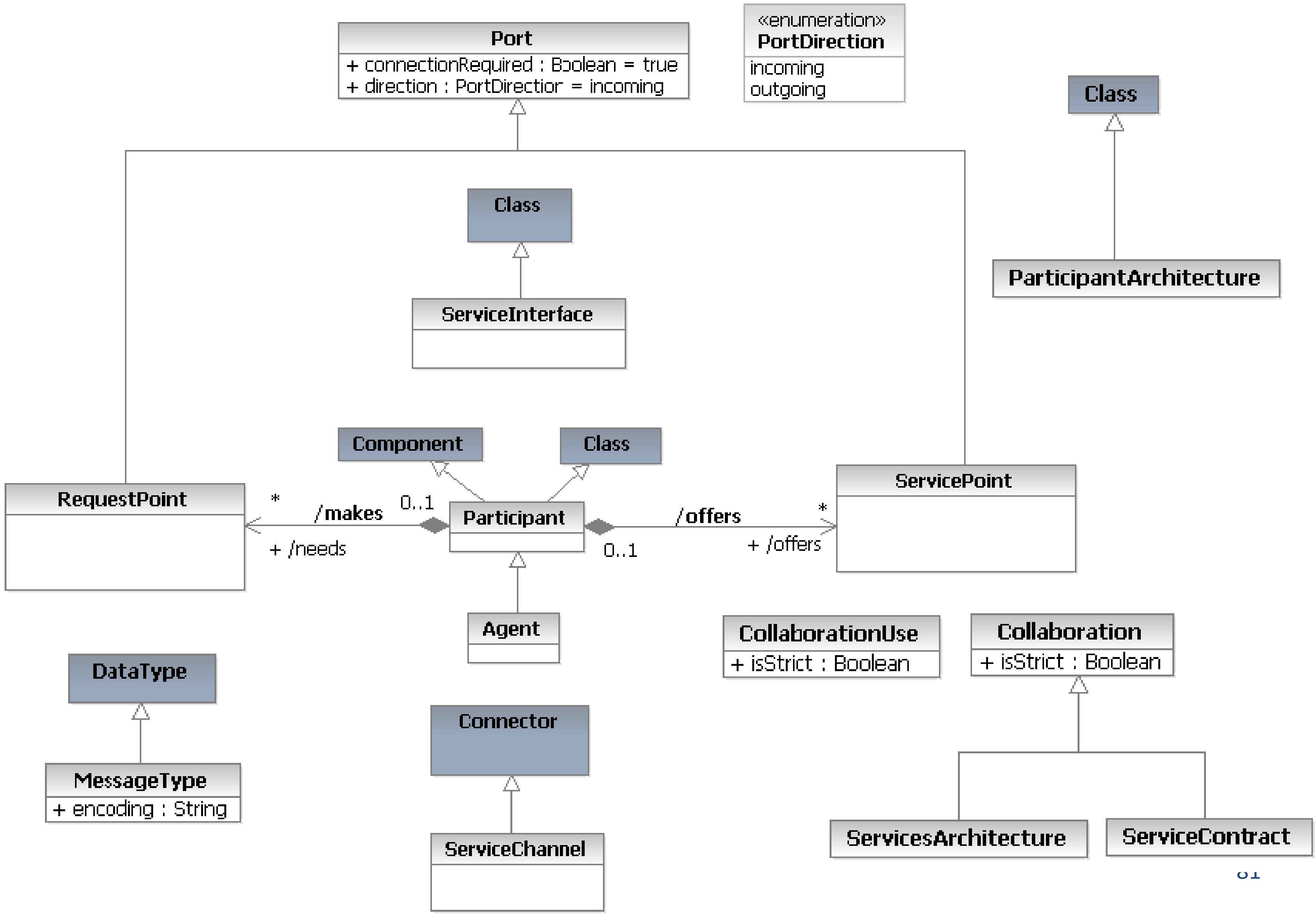


UML4SOA metamodel: Orchestration (cont.)

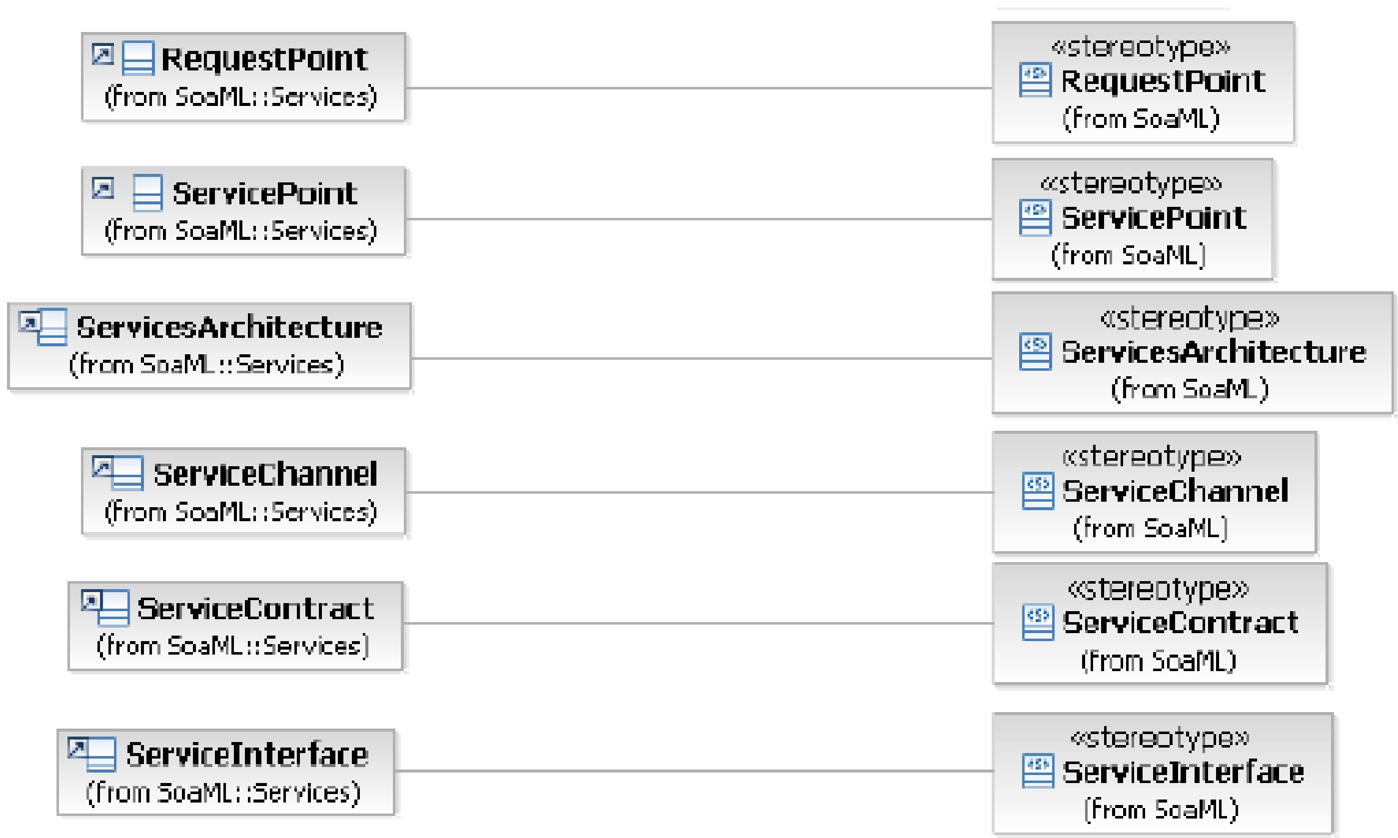
Conservative extension of the UML



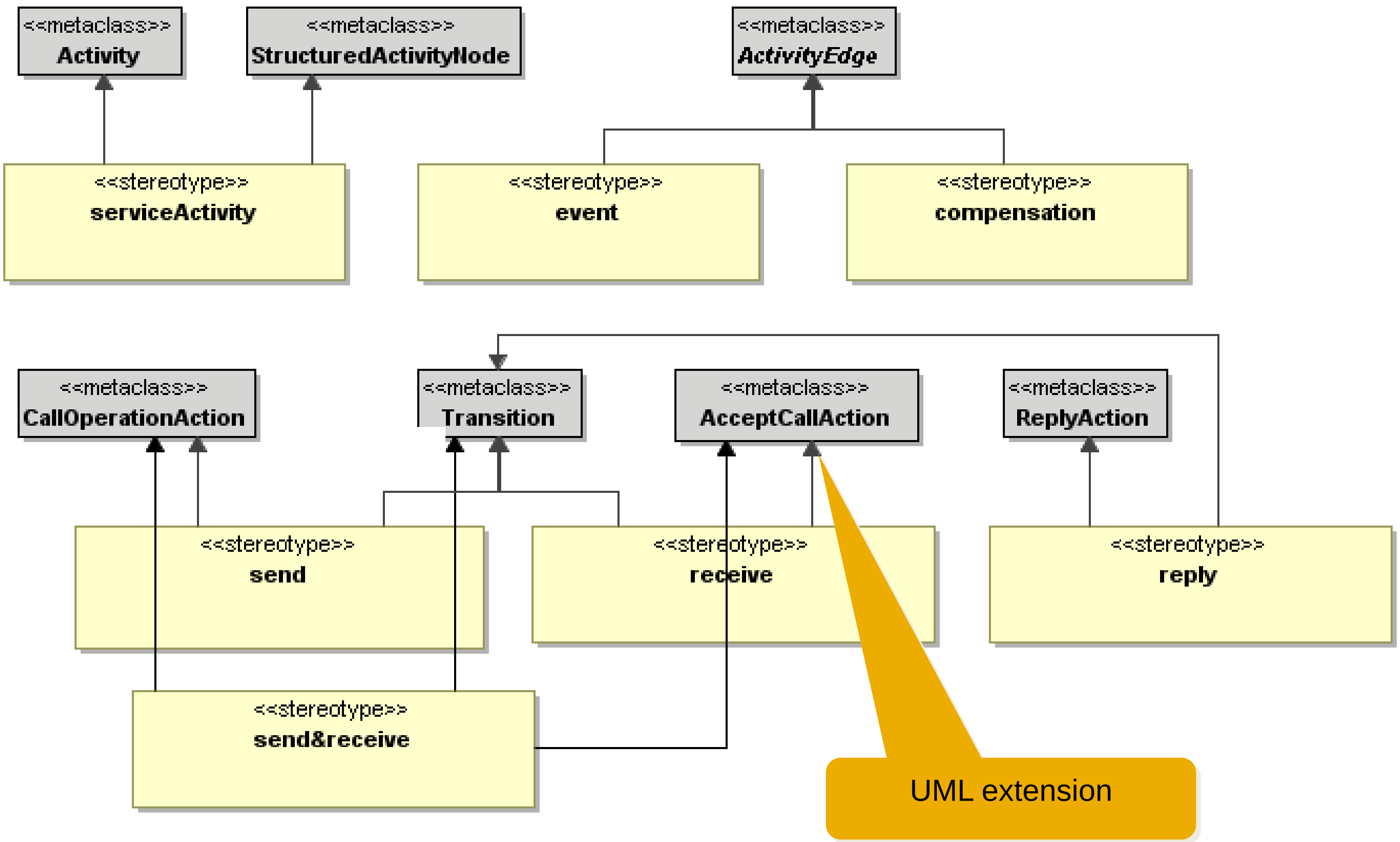
SoaML metamodel



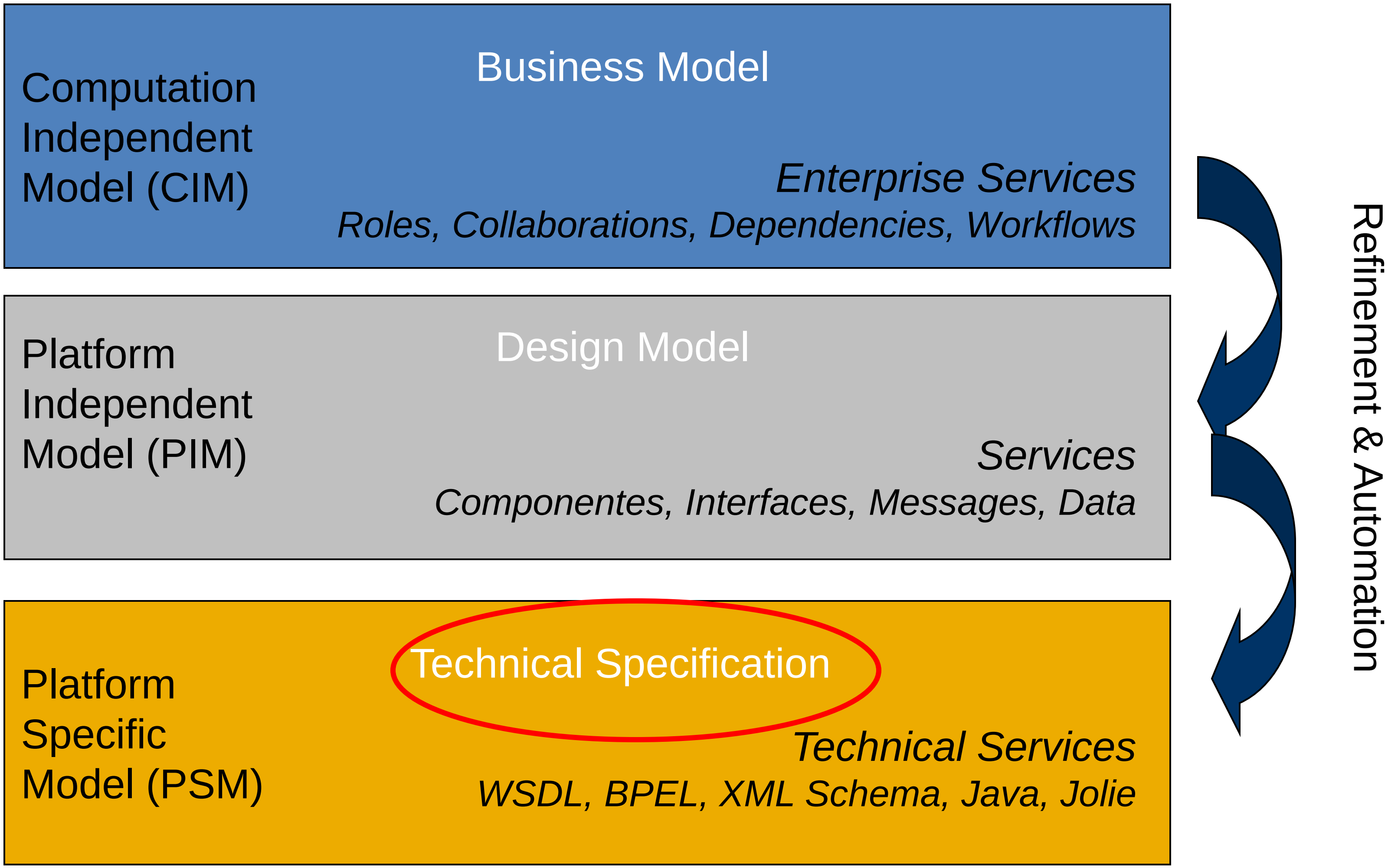
Profile metamodel mapping (excerpt)



Extension model (excerpt)



SOA models in the MDA context



Source: Data Access Technologies, Inc

Programming language Jolie

- Service-oriented paradigm
  - in Jolie everything is a service
  - used to create new services and compose existing ones
  - mechanisms for managing data, communication and service composition services
- Suitable for programming distributed applications
  - no distinction between local and remote services
  - endpoint locations and communication protocols can be changed dynamically thus allowing to build a dynamic system, fully reconfigurable at runtime

```
main {
    getInfo(request)(response)  {
        getTemperature@Forecast(request.city)(response.temperature)
        |
        getData@Traffic(request.city)(response.traffic)
    };
    println@Console("Request served!")()
}
```

service concurrently retrieves information from a forecast service and a traffic service:

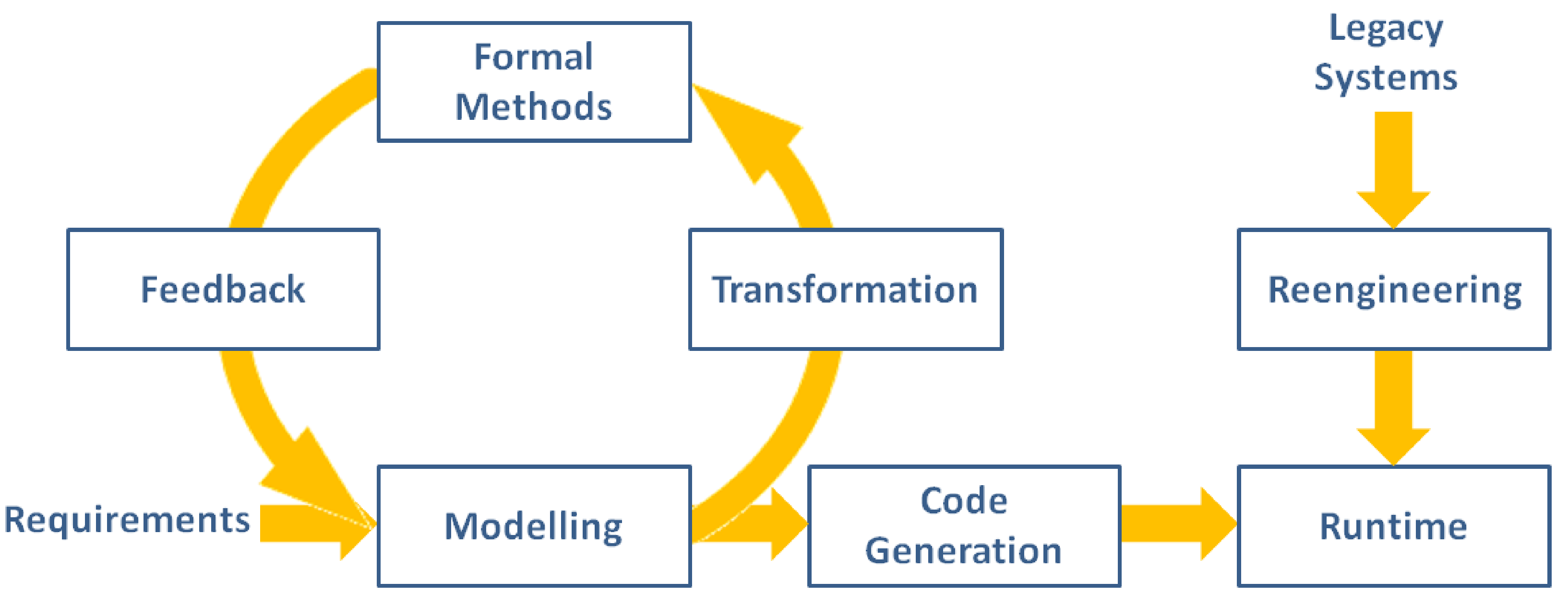
MDD4SOA

- MDD4SOA
  - Transformation mechanisms from models to executable orchestration of services
    - source: UML4SOA models
    - target platforms: BPEL/WSDL, Java, Jolie
    - fully automatic generation of code
    - implemented in Java

mdd4SOA

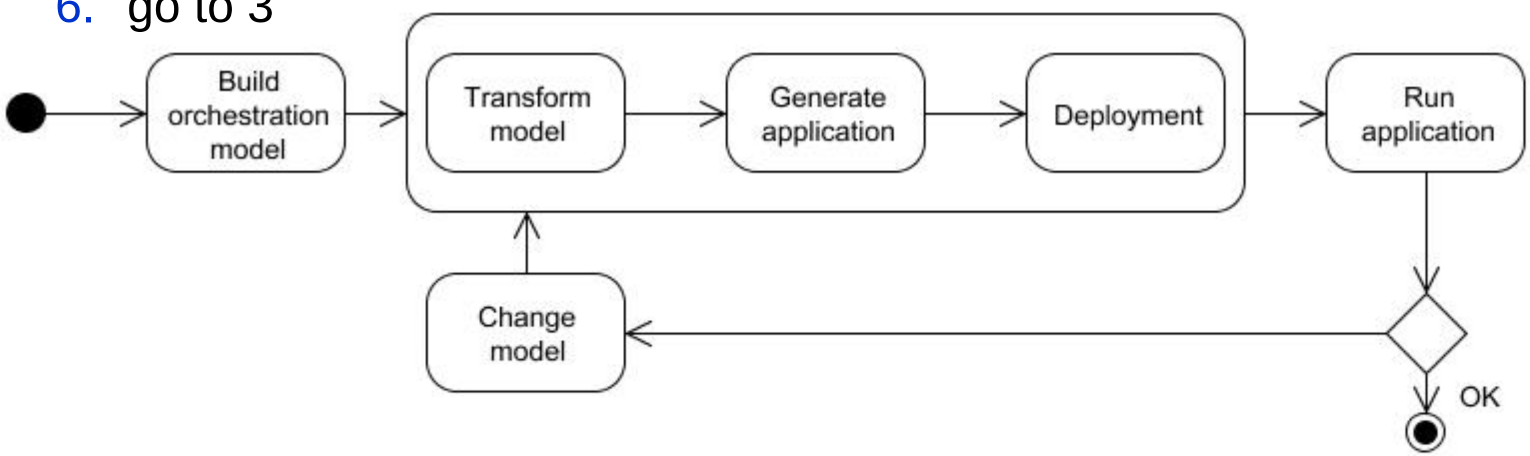
Mayer et al, EDOC 2008

Model-Driven Development@Work



MDD4SOA@work

- Demonstration's aim
  - to show how model-driven development of SOSs can work
- Consists of
  1. building an orchestration model with UML4SOA
  2. defining a tool chain of transformations in SDE
     - Analysis / model2model, model2code, deployment
  3. execution of the tool chain
     - input: UML4SOA model
     - output: application
  4. running the deployed application
  5. changing the model
  6. go to 3



SENSORIA Development Environment
(SDE)

Tools as services

- **Formal Analysis**
- **Transformation/Feedback**
- **Modelling**
- **Code Generation**
- **Runtime**

Tool Categories

SENSORIA Development Environment
(SDE)

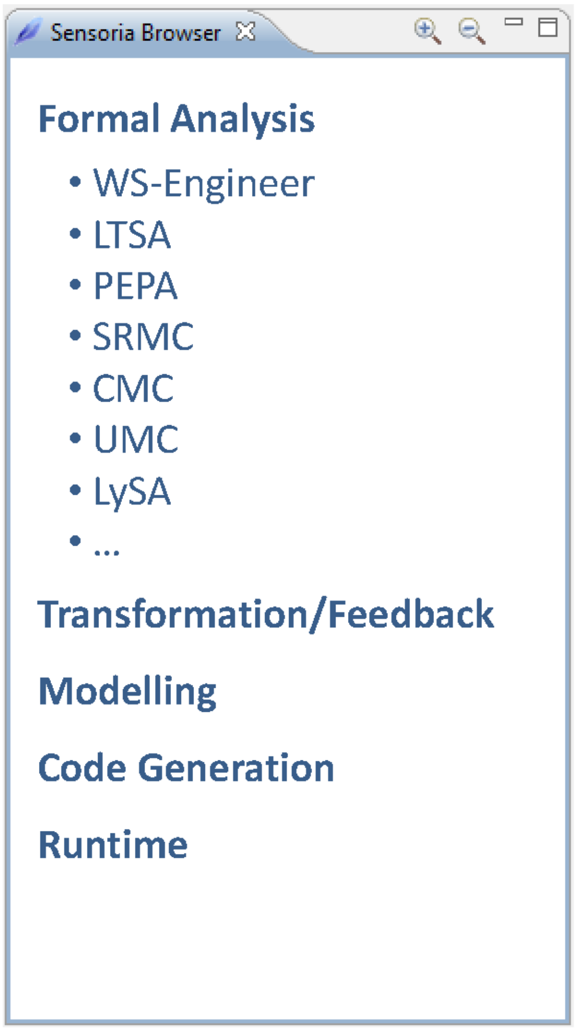
Tools as services

Tools

- **Formal Analysis**
  - WS-Engineer
  - LTSA
  - PEPA
  - SRMC
  - CMC
  - UMC
  - LySA
  - …
- **Transformation/Feedback**
- **Modelling**
- **Code Generation**
- **Runtime**

SENSORIA Development Environment (SDE)

Tools as services



SENSORIA: Over 20 tools in the SDE
NeSSoS: **Over 20 further tools** in the SDE

SENSORIA Development Environment
(SDE)

Tools as services

Sensoria Browser

Formal Analysis

- WS-Engineer
- LTSA
- PEPA
- SRMC
- CMC
- UMC
- LySA
- ...

Transformation/Feedback

Modelling

Code Generation

Runtime

UML2BPEL/WSDL Converter

UML to BPEL/WSDL Converter

convertToBPELWSDL()

...

Tool Info

WS-Engineer Interactions Check

WS-Engineer Interactions Check

convertToFSP()

checkFSPForSafety()

...

Tool Info

SENSORIA Development Environment (SDE)

Tools as services

Sensoria Browser

Formal Analysis

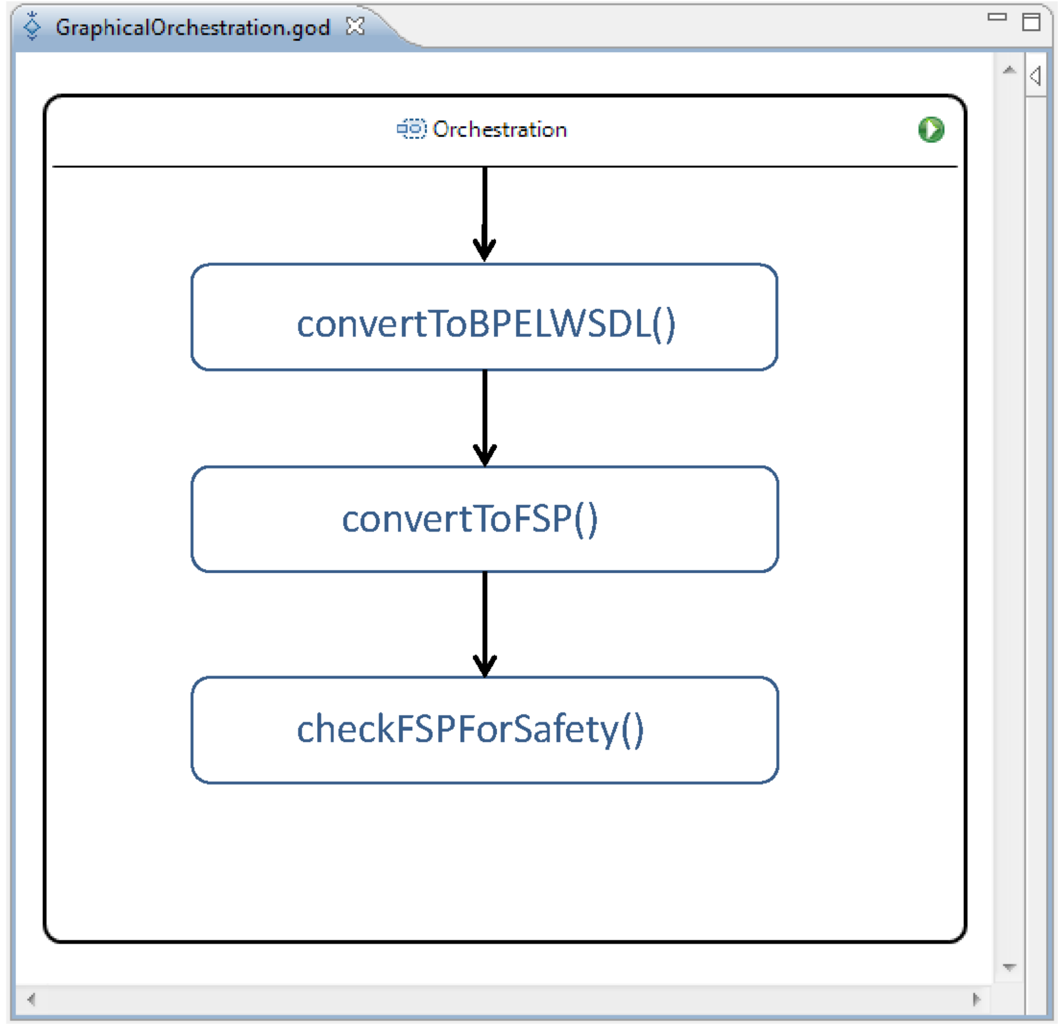
- WS-Engineer
- LTSA
- PEPA
- SRMC
- CMC
- UMC
- LySA
- ...

Transformation/Feedback

Modelling

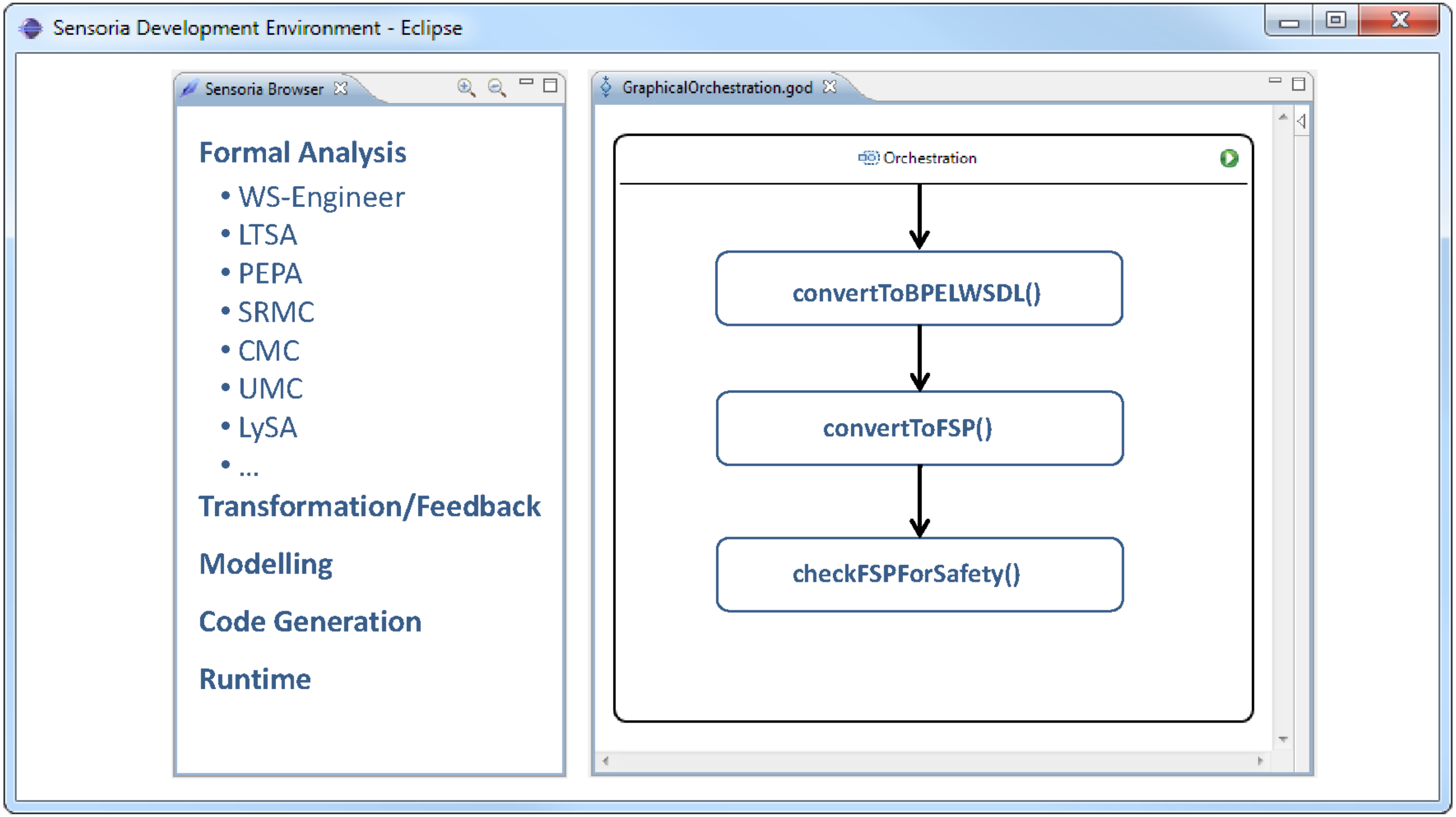
Code Generation

Runtime



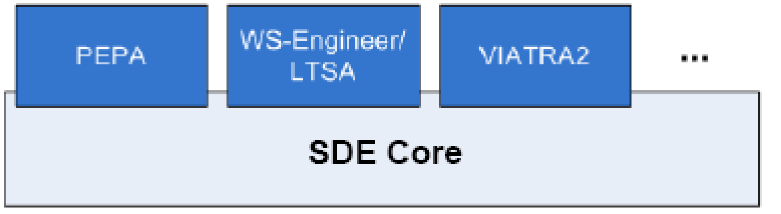
SENSORIA Development Environment
(SDE)

- **Integration into Eclipse**



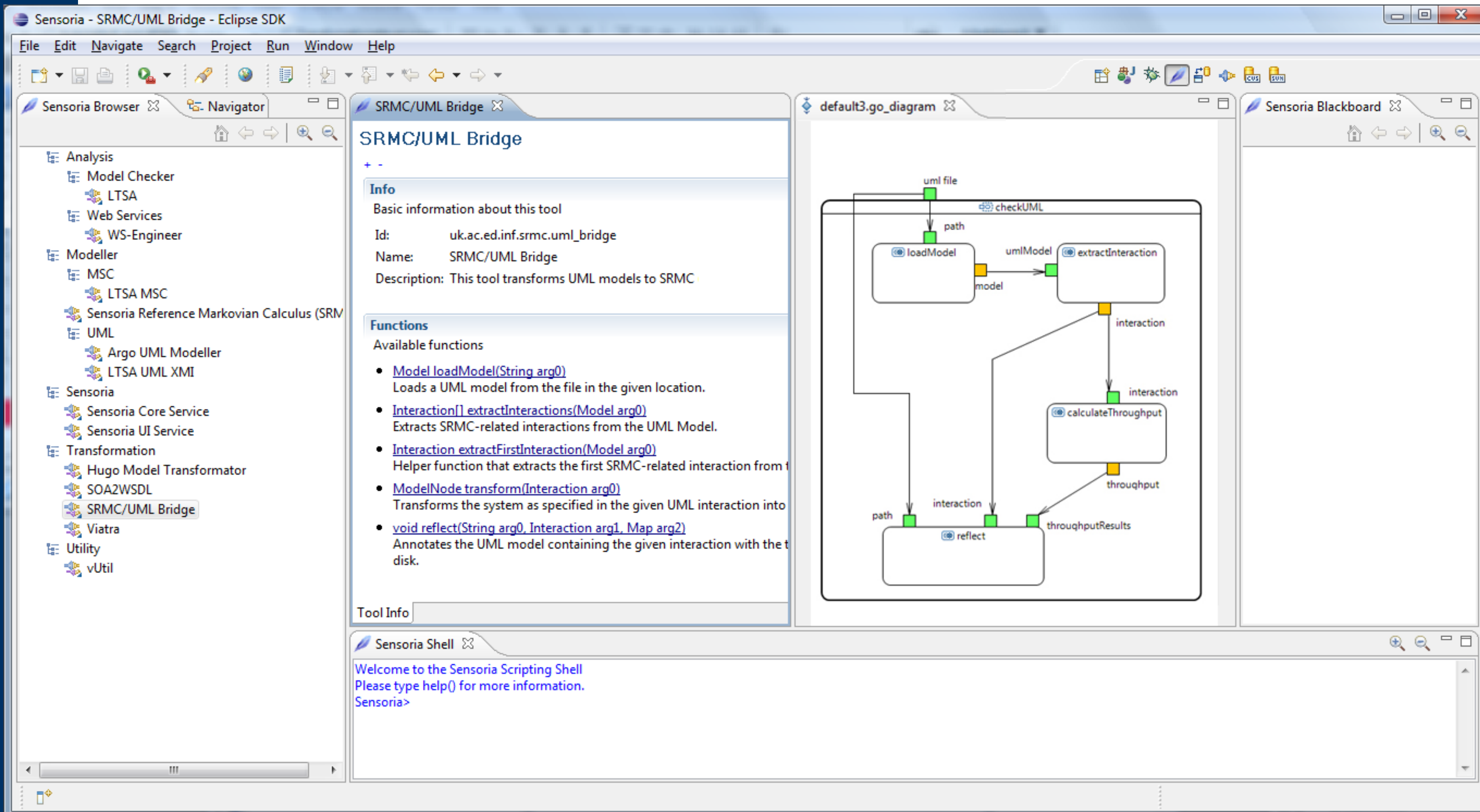
SENSORIA Development Environment
(SDE)

- Eclipse-based integration platform for developing SOA-based software
  - SDE Core
  - integrated tools



- Distinctive features of the SDE Core
  - uses a SOA approach itself
  - tools are orchestrated by the specification of a tool chain
  - tool-as-service concept: Orchestrations of tools are now usable as tools themselves
  - enables SOA developers to use tools without the need to understand the underlying formal languages
- Tool chain in SDE
  - defined as a SDE script
  - drawn with the graphical orchestration tool
  - executable in the Eclipse environment

SDE (Sensoria/Service Development Environment) (contact Philip Mayer)



http://svn.pst.ifi.lmu.de/trac/sde

SDE

See short film

Selection of tools, techniques, methods, languages, …

- SENSORIA approach, in particular the integrated tools in SDE encompasses
  - the whole development process of service-oriented software
  - from systems in high-level languages to deployment and re-engineering
- **Difficulty to identify the "best" techniques and tools (SDE plug-ins)**
  - for solving a particular problem arising in the development process
- To ameliorate this problem we defined a catalogue of patterns
  - serves as an index to our results
  - illustrates, in a concise manner, the advantages and disadvantages of the individual techniques

Example: Service modelling pattern
(simplified description)

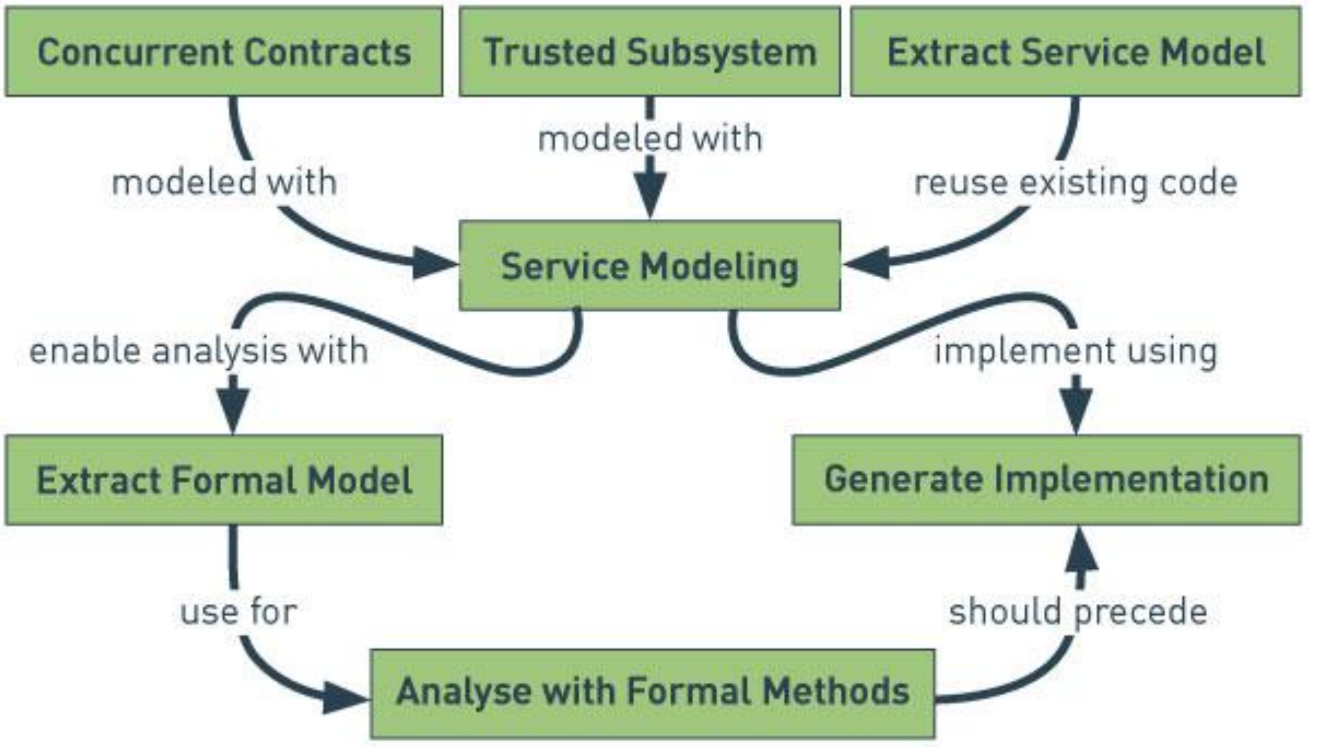
- Context
  - you are designing a SOA-based system
  - the system is intended to offer services to multiple platforms and makes use of existing services on multiple hosts
- Problem
  - when designing SOA systems, it is easy to get lost in the detail of technical specifications and implementations
  - need of effective task identification, separation, and communication
- Forces
  - amount of specifications and platforms in the SOA domain makes it difficult to get a general idea of the solution space
  - having a global architectural view eases the task of understanding the SOA environment

Example: Service modelling pattern (cont.)
(simplified description)

- Solution
  - use a specialised (graphical) modelling language to model the system
  - employ these models as far as possible for generating the system implementation
- Consequences
  - Pros: better idea of how the individual artifacts fit together and better communication between developers and customers
  - Cons: Often fully automated generation of code is not feasible
- Tools
  - UML CASE tools (Rational Software Modeler, MagicDraw, …)
    - profiles SoaML, UML4SOA
  - SENSORIA Development Environment (SDE)
    - model transformations MDD4SOA
- Related patterns
  - Extract formal models
  - Generate implementation

Pattern catalogue

- Relationships between patterns

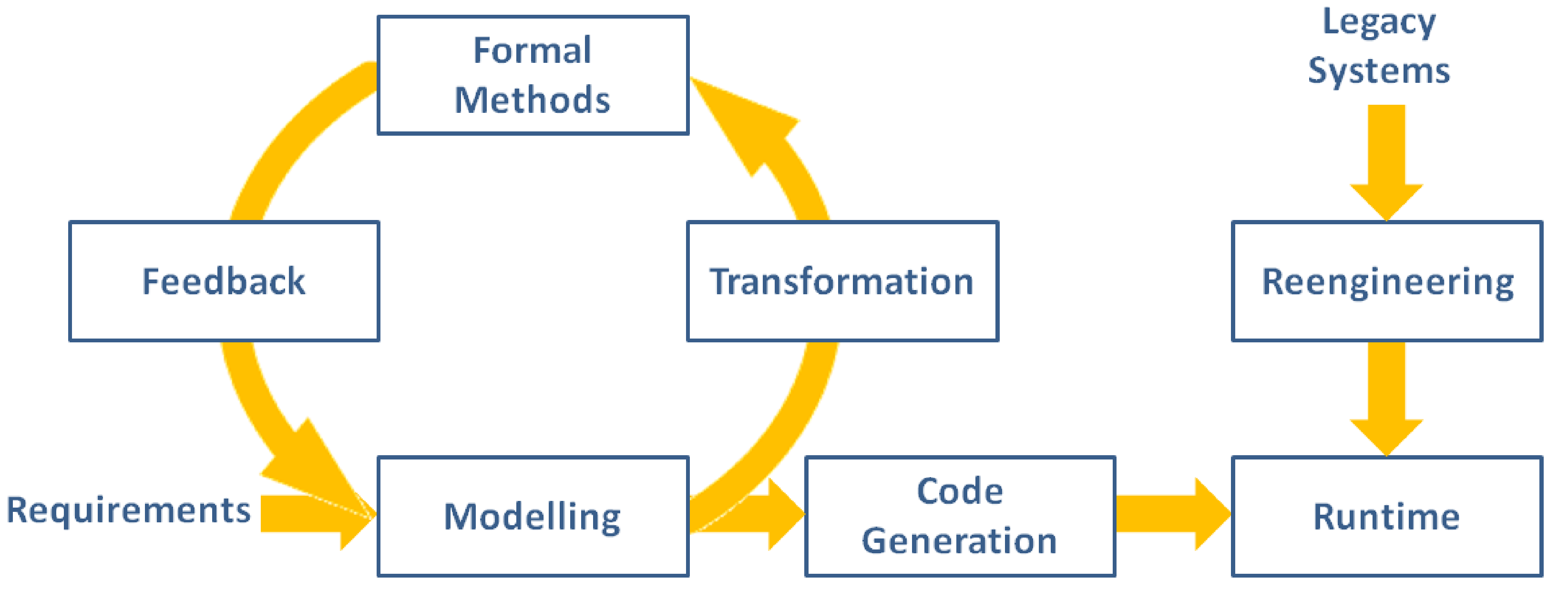


Case Study
Automotive scenario

- Scenario On Road Assistance
  - Driver is on the road with his car
  - Diagnostic system reports a low oil level; the car is being no longer driveable
  - Driver contacts the on road assistance system
  - Car position is located
  - System finds appropriate services in the area (garage and rental car)
  - Based on the drivers preferences the best services are selected
  - Driver is required to deposit a security payment by credit card
- On Road Assistance as orchestration of services
  - services: car position, finding garage and car rental station, selection of best service, charge credit card
- Application: visualisation of invoked services
  - Each service has associated a user interface (web page)

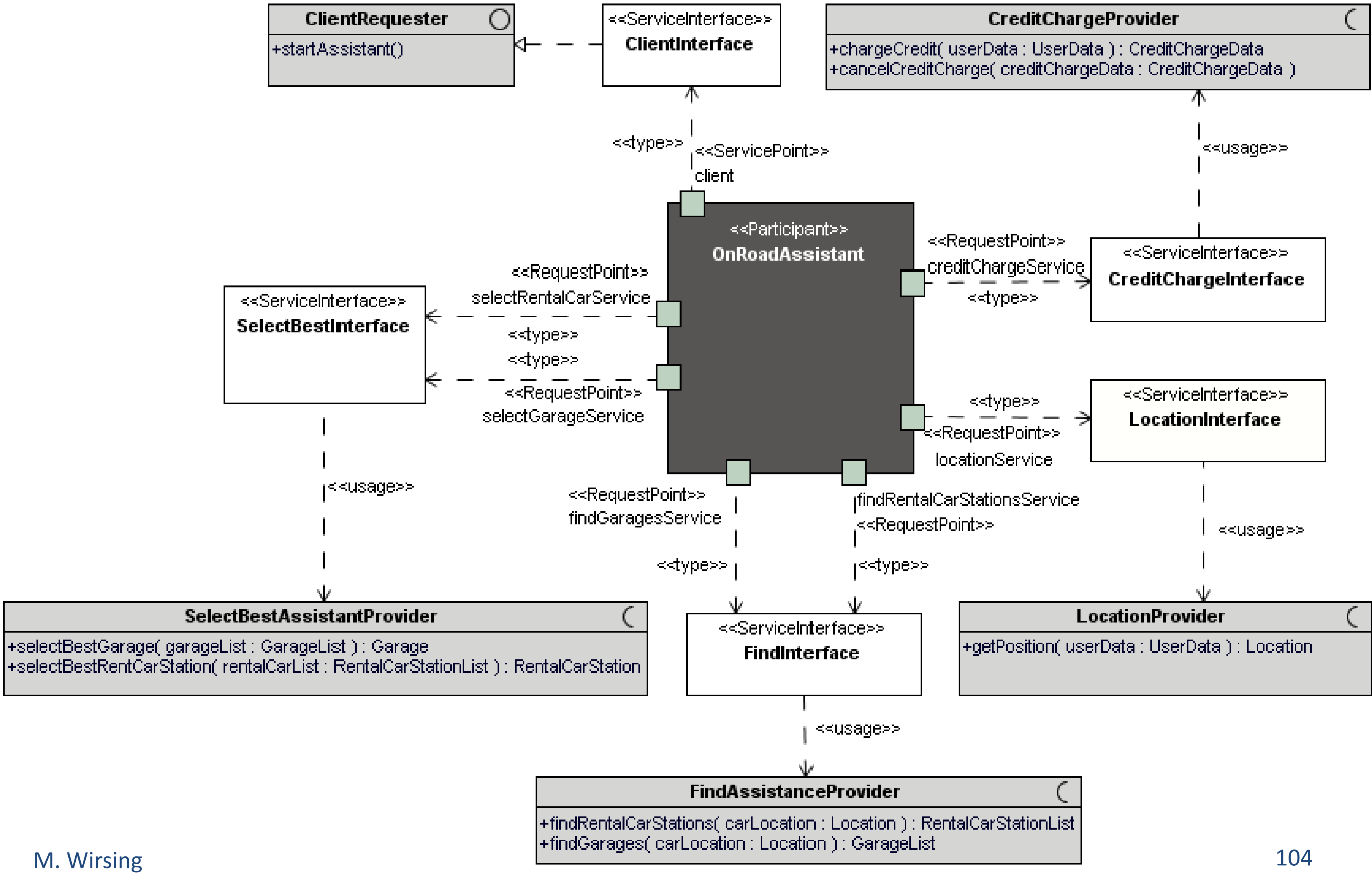
SOA Development Process (recap)

1. Construct and validate business model (requirements)
2. Build design model
3. Analyse properties and refine design model
4. Generate SOA realization



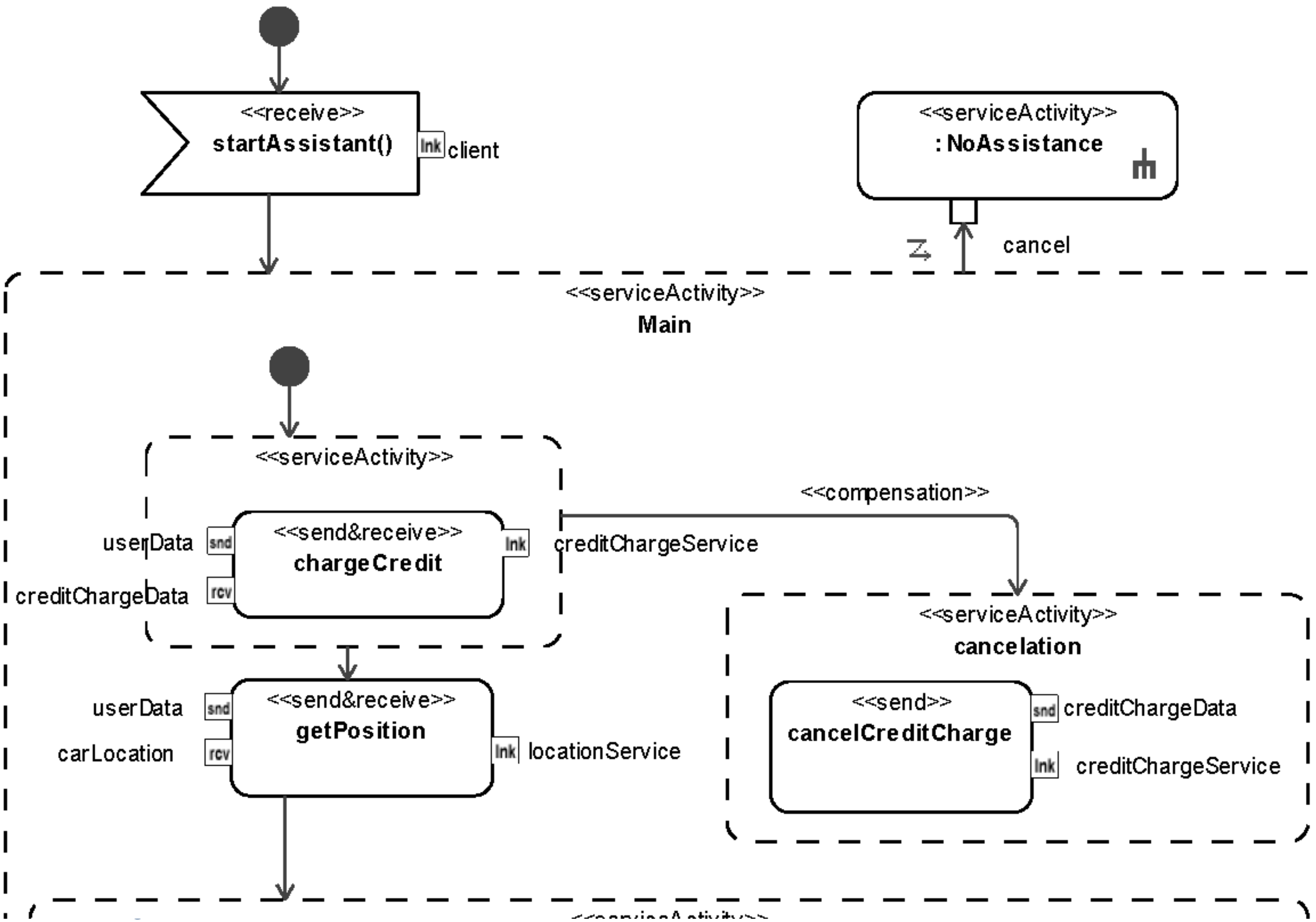
1. Design model (static structure)

On Road Assistance scenario



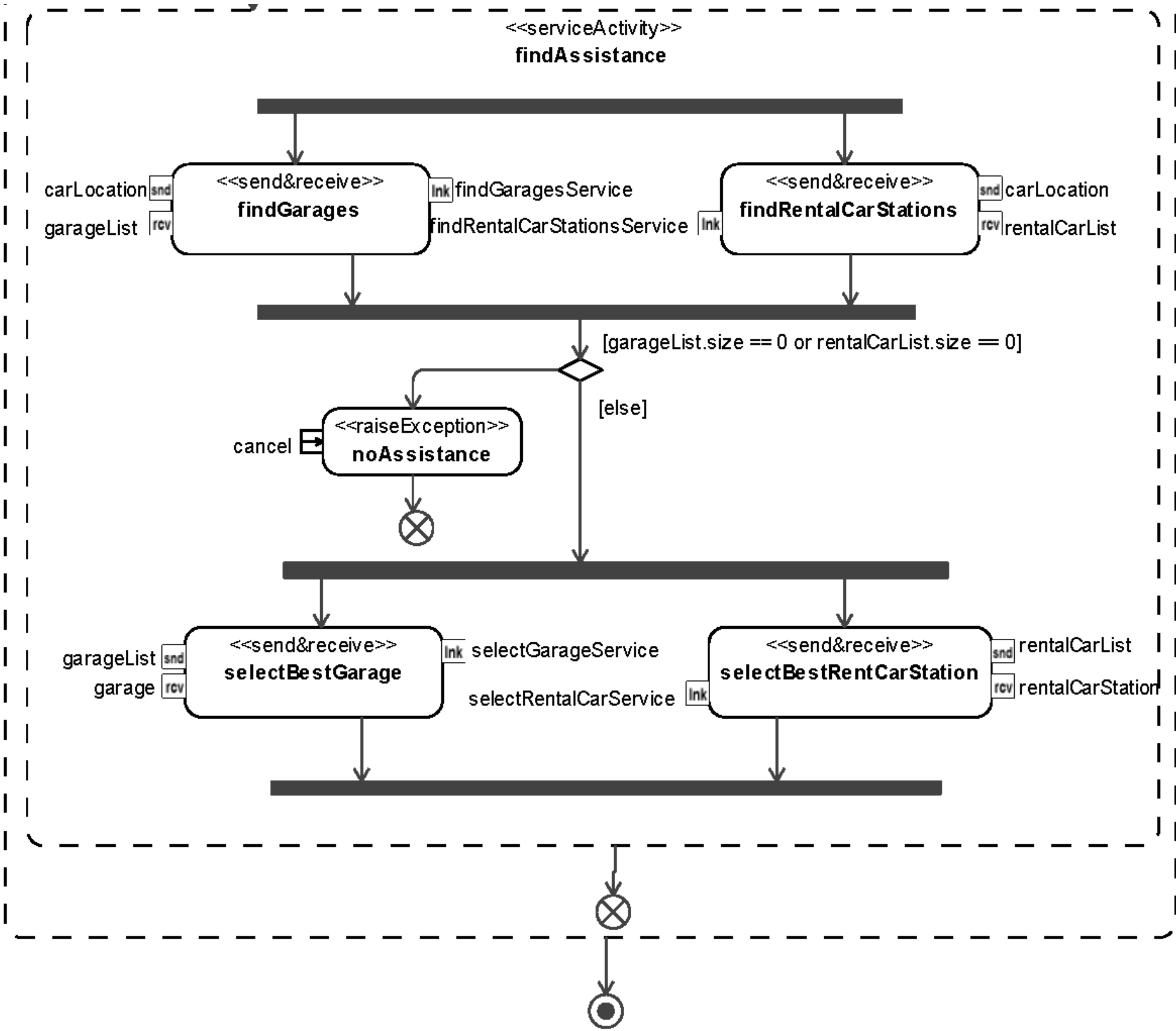
1. Design model (orchestration)

On Road Assistance scenario



1. Design model (orchestration, continued)

On Road Assistance scenario



2. Selecting the „Best“ Service

- The SelectGarageService computes a list of best offers according to user constraints and preferences, e.g.
  - **Fast repair:** Repair as soon as possible, in less than 48 hours
  - **Preference:** Prefer fast repair to cheap repair
- **SENSORIA Approach**:
  - **Soft Constraints over C-Semirings** [Bistarelli, Montanari, Rossi 97]
  - **Policy language with preferences** [W, Hölzl 06]
- **Idea**:

  Solve optimisation problems abstractly over constraint semirings
- A soft constraint C is given by
  - A **(finite) set X of problem variables** over a domain D
  - A mapping of type

    (X -> D) -> S

    which assigns values in a **semiring S** to valuations of X

Soft Constraints and Preferences for Services

Soft constraint system for choosing the „best“ offer

- **Variables** garage-cost, garage-duration, …
- **Domain** D = { n ε N: 0 <= 10000 }
- **Semiring** FuzzyRing = <R^+, max, min, 0,*1*>

Soft Constraints and Preferences for Services

Soft constraint system for choosing the „best“ offer

- **Variables** garage-cost, garage-duration, …
- **Domain** D = { n ε N: 0 <= 10000 }
- **Semiring** FuzzyRing = <R^+, max, min, 0,*1*>

- **Constraints and preferences**
  - **Repair as soon as possible, in less than 48 hours**

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

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

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

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

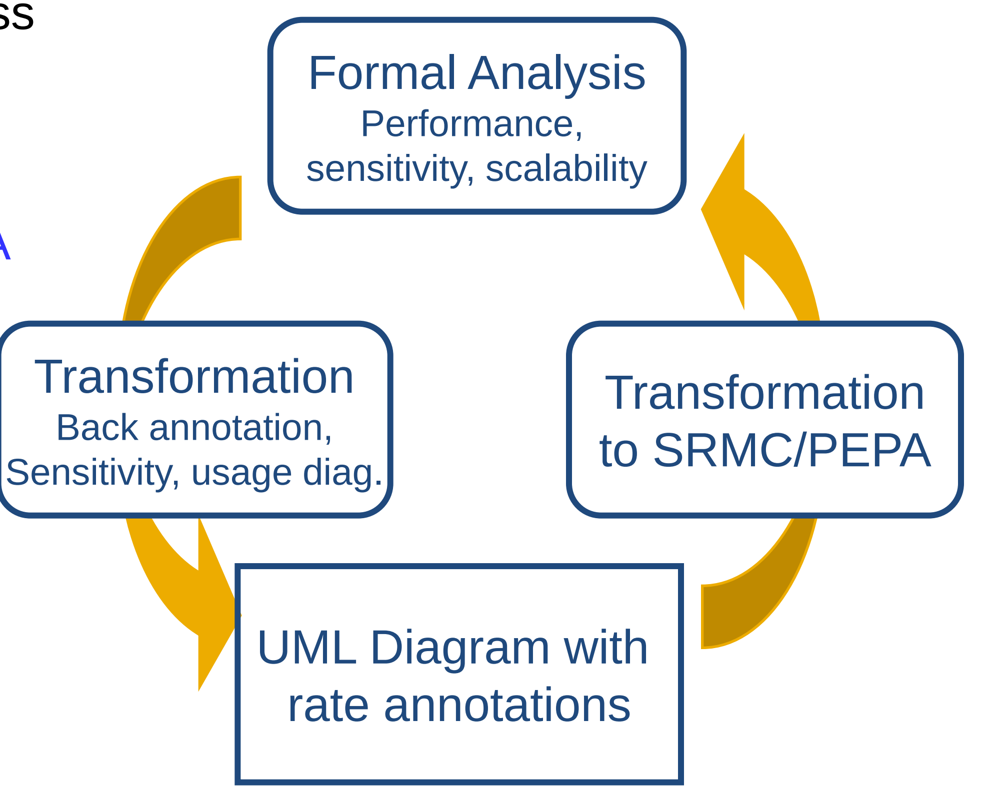
    fastRepair > cheapRepair

3. Analysis of Quantitative Properties:
Service Level Agreements

- Specifying performance by annotating UML diagrams & translation into stochastic process calculus PEPA
  [DEGAS Project 2004]

- Extension to SRMC (SENSORIA Reference Markovian Calculus)
  [Gilmore et al. 2006]

- Performance, sensitivity and scalability analysis of Service Level Agreements
  using
  - Continuous Markov chains
  - Ordinary differential equations [Gilmore, Hillston 2005]
  - Parameter sweep [Gilmore et al. 2006, 2007]



Example:
Performance of Road Assistance

- Can we guarantee the following Service Level Agreement?

  At least 30% of engine failures lead to garage and rental car being **ordered within fifteen minutes** and

  at least 60% of engine failures lead to garage and rental car being ordered within thirty minutes.

- Approach:
  - **Add rates to the time-consuming actions of the workflow**
  - **Translate activity diagram to SRMC**

Transformation to SRMC

- The Road Repair System (simplified)

```
OnRoadAssistant ||_L
 (LocationSvc || FindGrgeSvc || FindRentalCarSvc
  CChargeSvc || SelectGrgeSvc || SelectRentalCarSvc)
```

- Determining the current location of the car and finding nearby services:

```
OnRoadAssistant = (start,r0).
    (chargeCredit, infty).(getPosition, infty).
    ((findGarage,infty) || (findRentCarStation, infty)).
     OnRoadAssistant1
LocationSvc = (getPosition, r2). LocationSvc  ...
```

Passive waiting, not determining the rate

0.9 .. 1.1; locaton info can be transmitted in 1 min, with little variance

- Selecting garage and rental car

```
OnRoadAssistant1 = ((selectBestGarage,
    (selectBestRentalCar, infty)). OnRoadAssistant
SelectGrgeSvc = (selectBestGarage, r5). selectGrgeSvc
```

0.15 .. 1.0; processing orders may take 5 min, with high variance

Analysis of Service Level Agreements

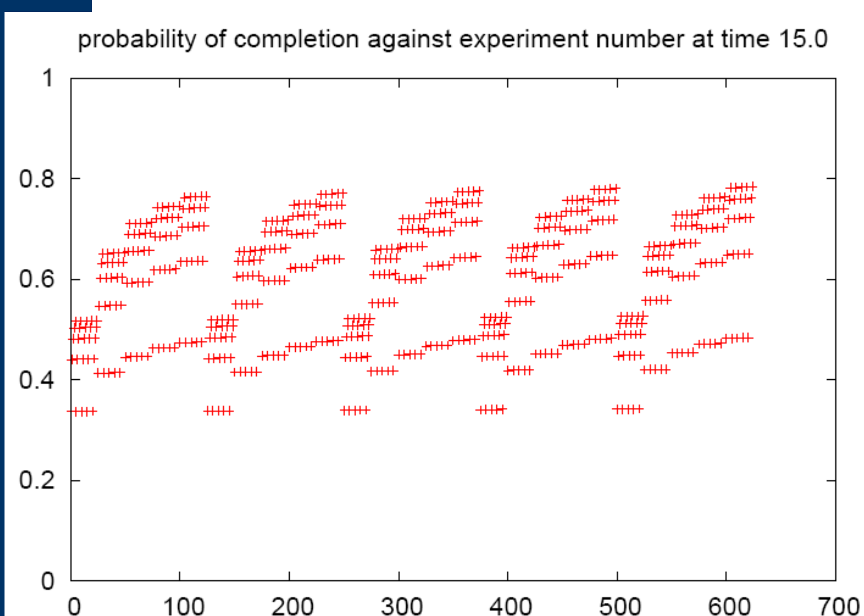
- **Example Service Level Agreement:**

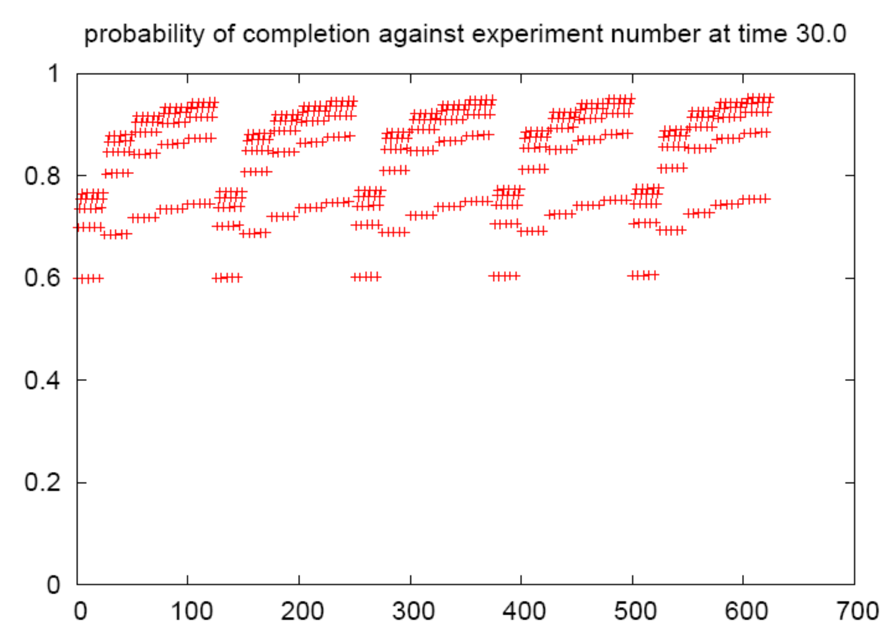
  At least 30% of engine failures lead to garage and rental car being ordered within fifteen minutes and

  at least 60% of engine failures lead to garage and rental car being ordered within thirty minutes.

- **Analysis by varying rates r1-r5:**

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

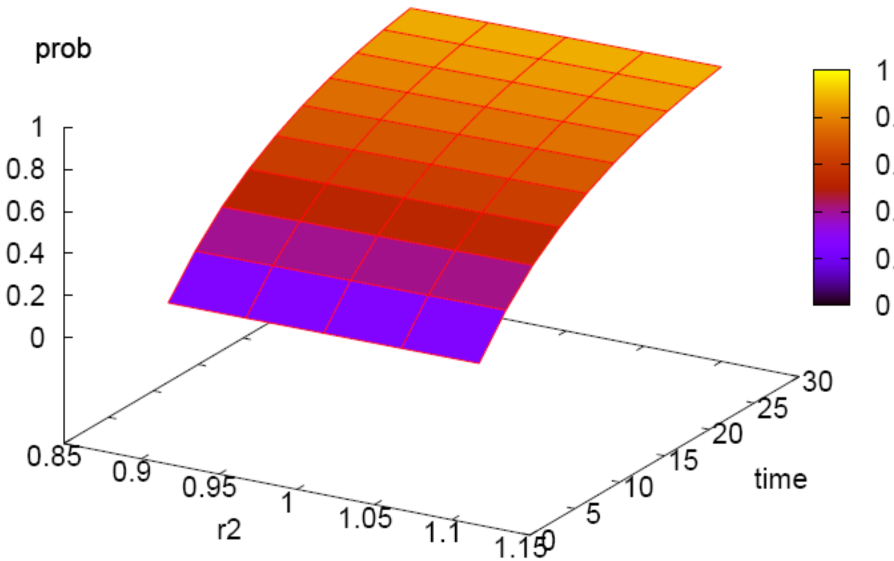




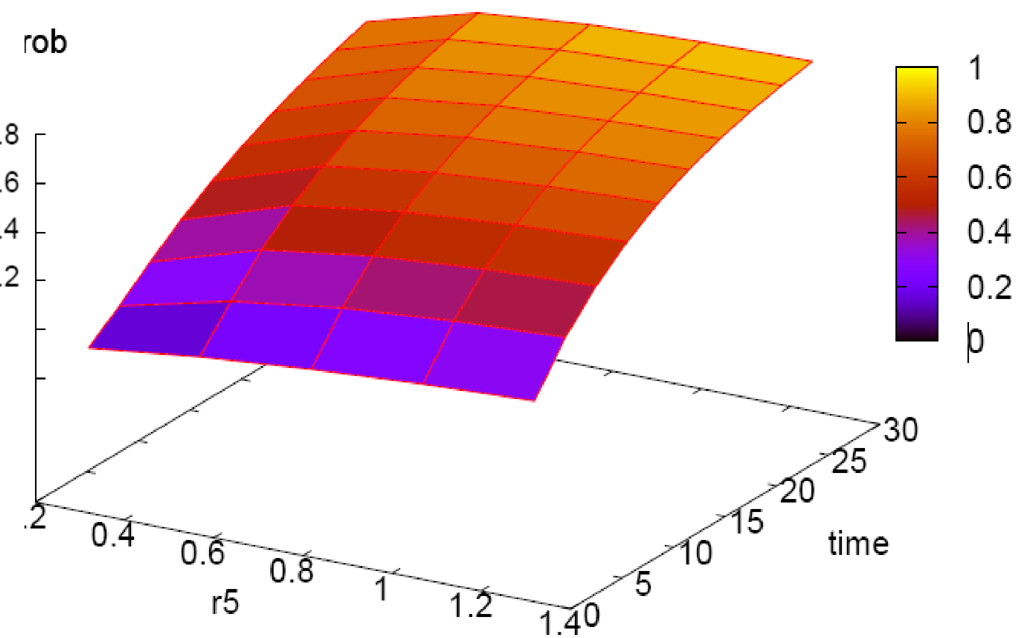
Analysis of Service Level Agreements

- Cumulative analysis of Service Level Agreement:

Sensitivity to variation of r2



Sensitivity to variation of r5



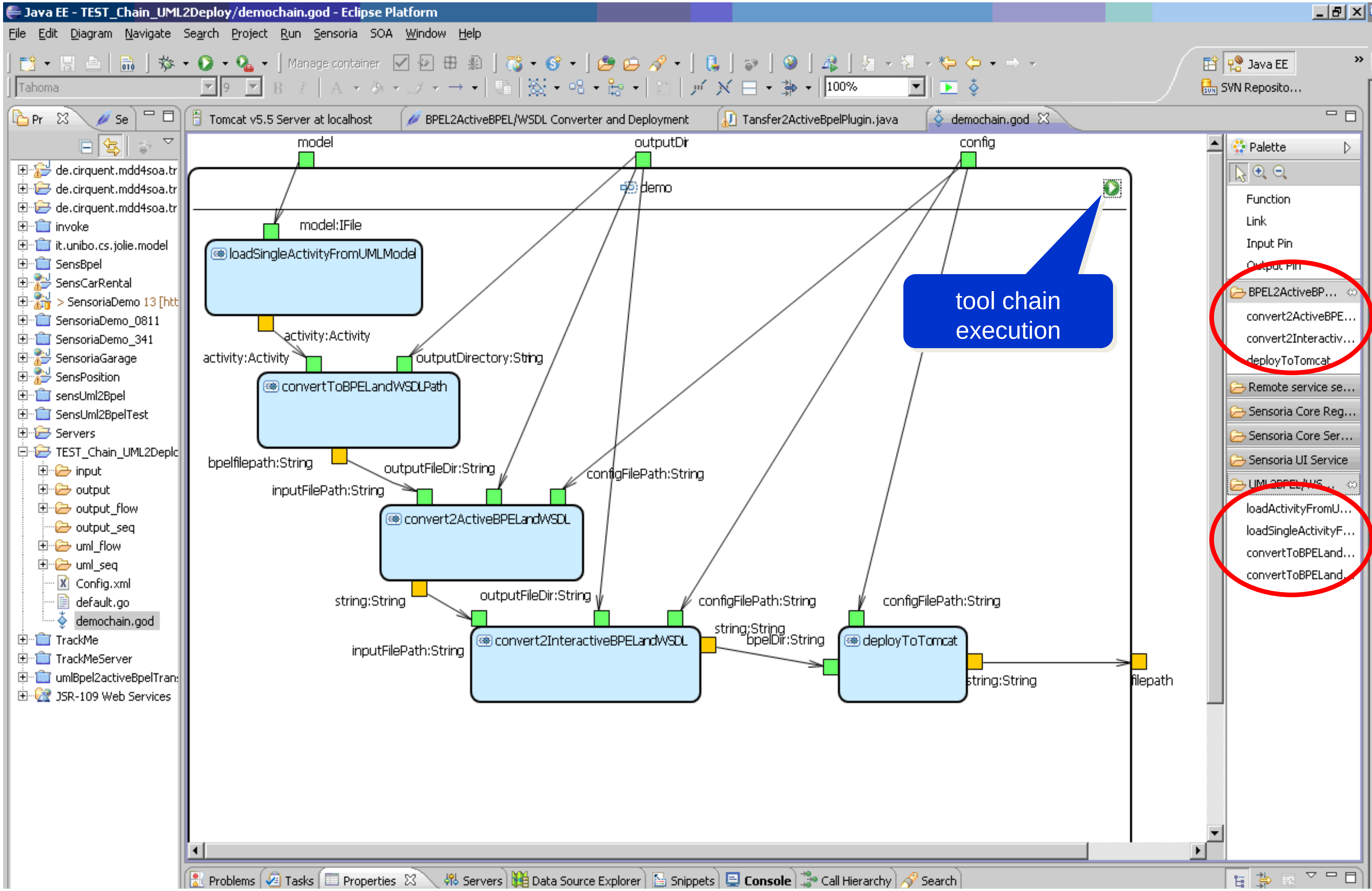
Consequence: A faster processing time for orders (governed by rate *r*5) is more important than trying to transmit location data faster (governed by rate *r*2).

4. Defining tool chain in SDE

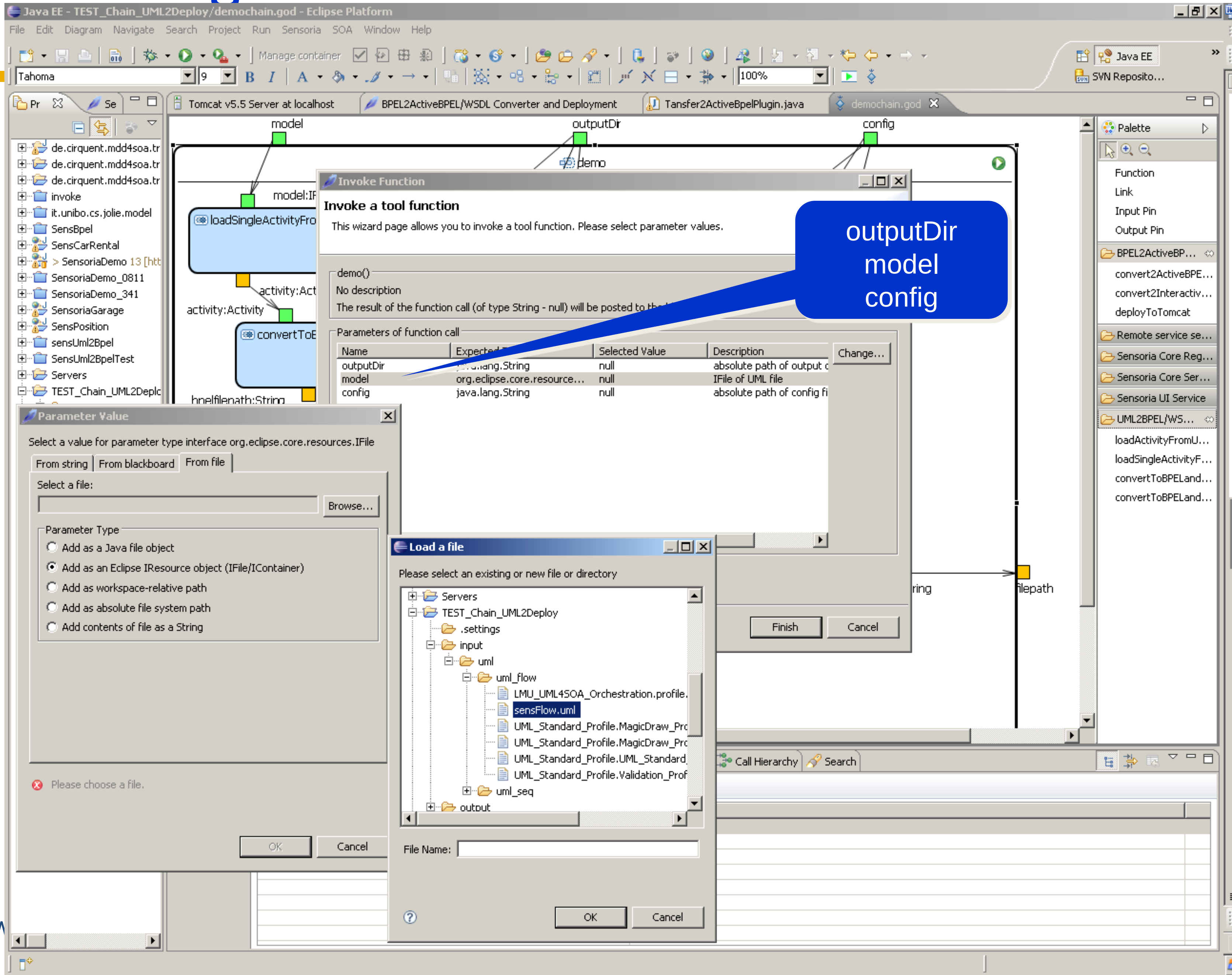
- Converter UML4SOA to BPEL/WSDL
  - transformation from UML2 models to an Intermediate Orchestration Model (IOM)
  - transformation from IOM to BPEL/WSDL*
- Converter BPEL/WSDL to active BPEL/WSDL
  - transformation of BPEL/WSDL* to code executable by ActiveBPEL Engine 4.0 (open source)
    - Replacement of namespace and service location within BPE /WSDL
    - Create process deployment description files (catalog.xml, *.pdd)
- Transformation active BPEL to interactive BPEL
  - transformation for adding user interaction mechanisms
    - additional *receive* & *reply* for each *invoke* for communication between user and BPEL process
    - extension of *reply* with a list of next actions
- Deployment on a web server (Tomcat)

Tool chain in SDE

Graphical orchestration of tools (Eclipse plug-ins)

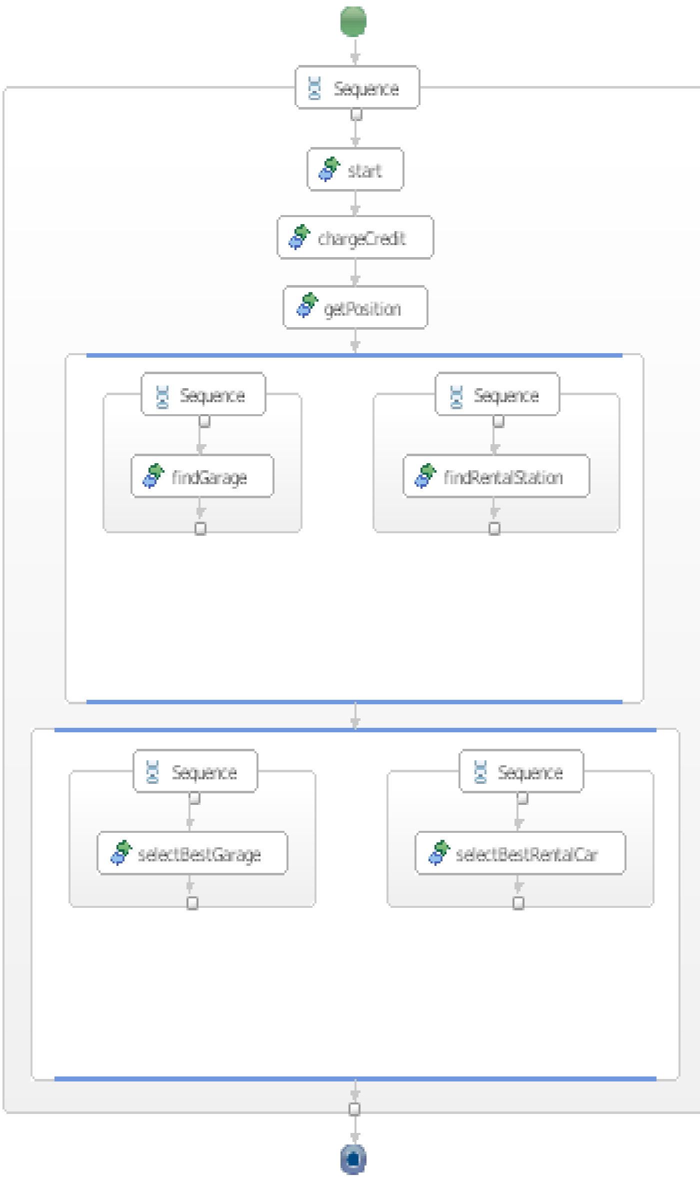


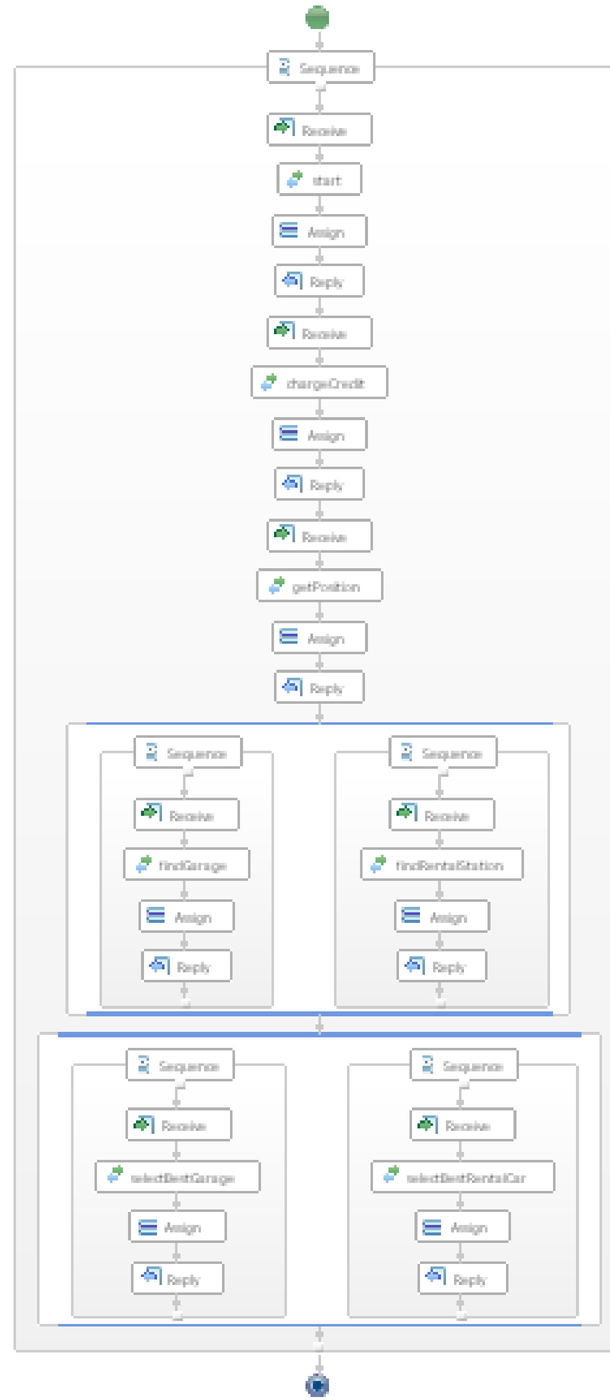
5. Executing tool chain



Looking at transformation results

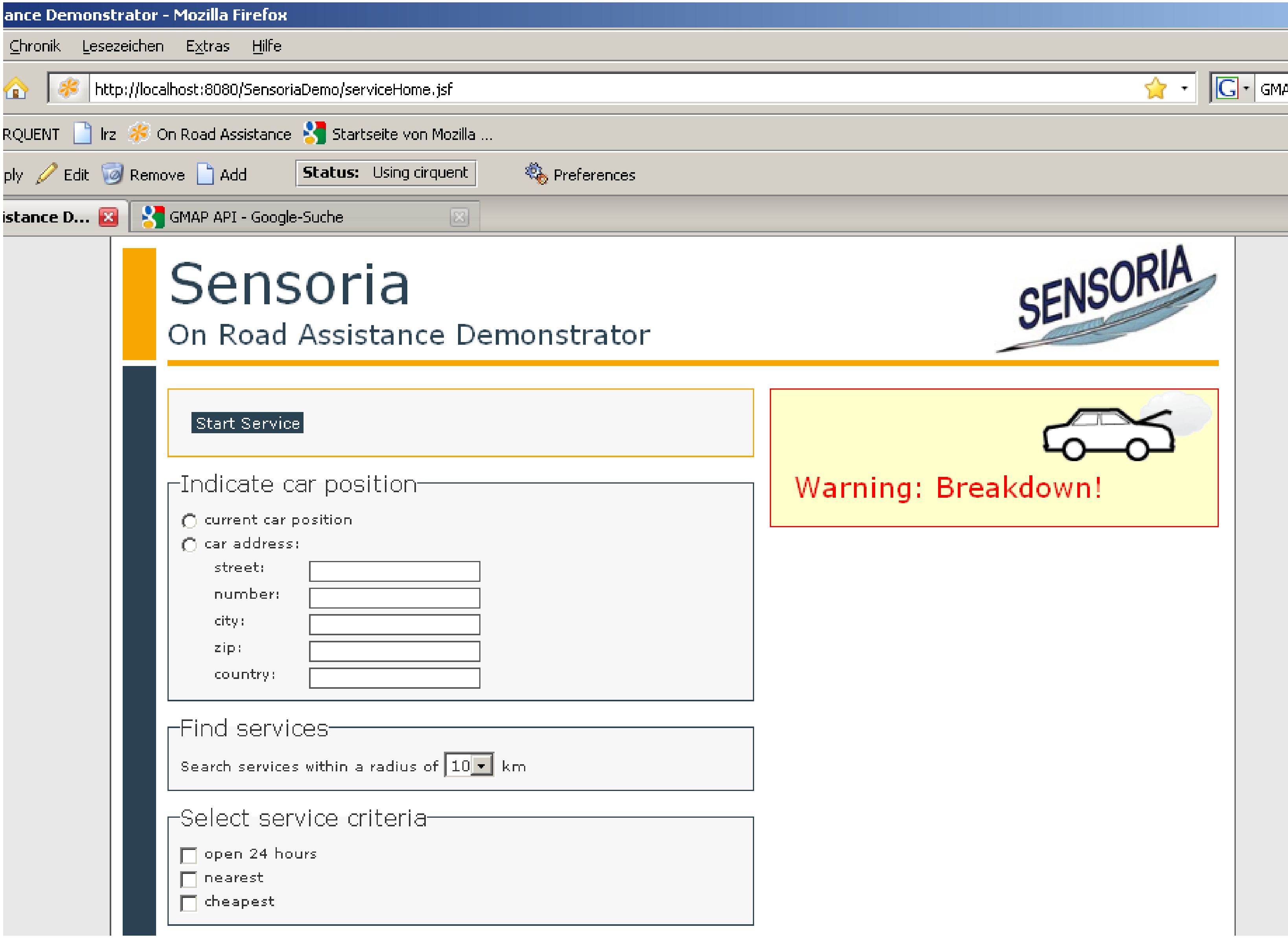
BPEL models





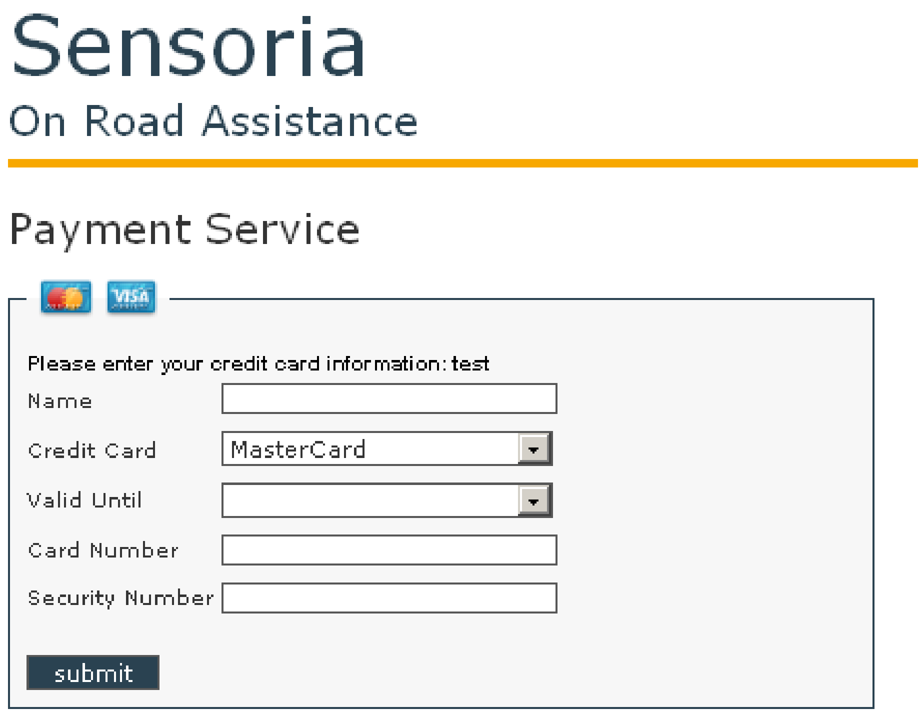
6. Running the deployed application

Home Page - Setting of Preferences



6. Running the deployed application

Credit card charge



6. Running the deployed application

Car position

Sensoria

On Road Assistance

SENSORIA

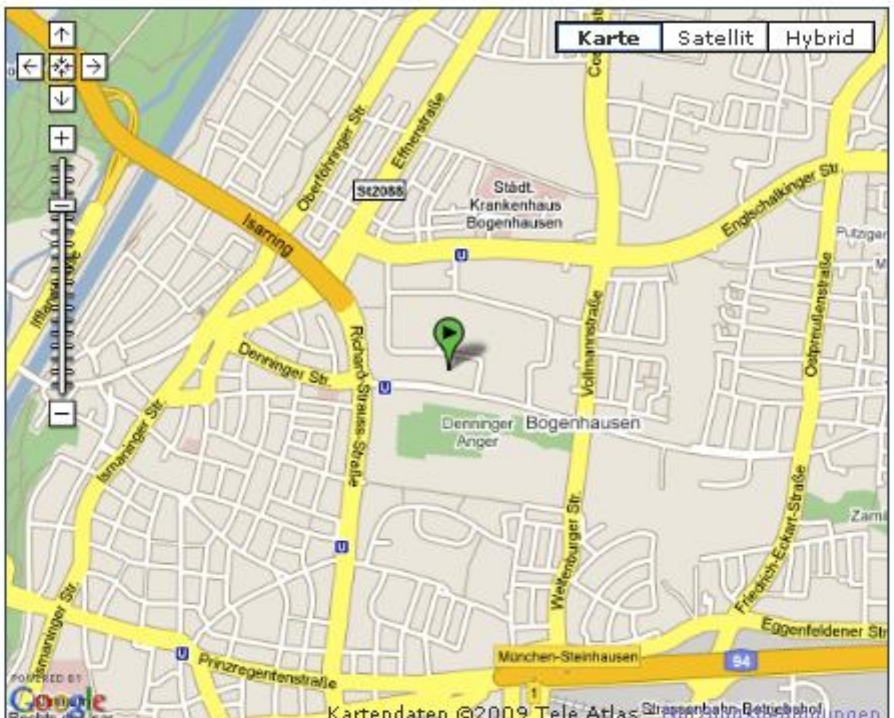
Car Location

Next step

search rental car station nearby
search garage nearby

continue

Current Location



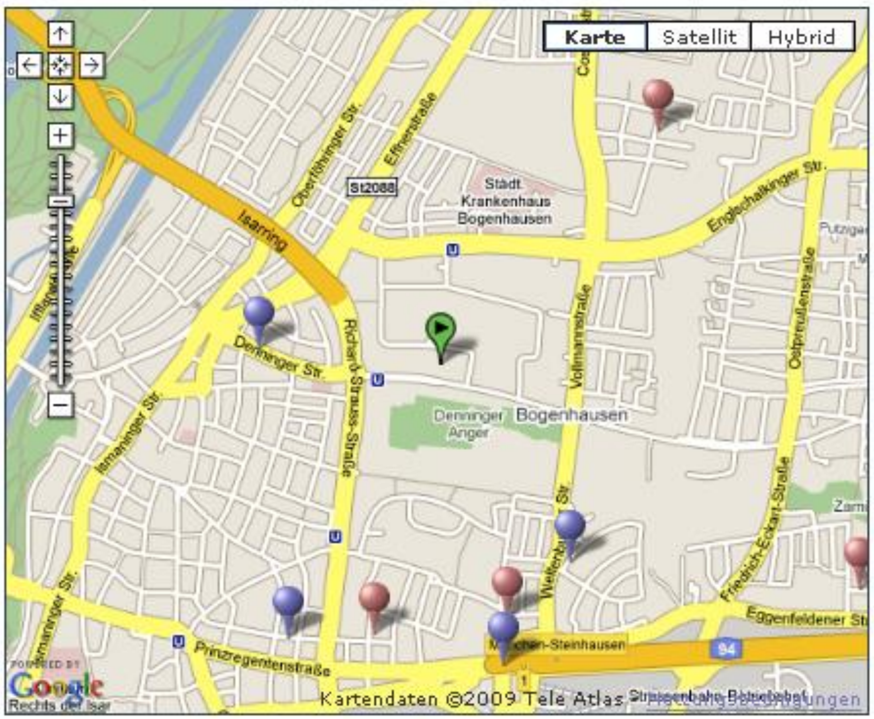
6. Running the deployed application

Garage and rental car services

Sensoria

On Road Assistance

Garage nearby your car
rental car station nearby your car



Next step

search best garage
search best rental car station

continue

Current Location

Garage nearby your car

Garage Denninger	get route
Garage Neckar	get route
Garage Riedenburger	get route
Garage Zaubzer	get route

rental car station nearby your car

Car Rental Gotthelf	get route
Car Rental Steinhauser	get route
Car Rental Eva	get route
Car Rental Ina	get route

6. Running the deployed application

Selection best services

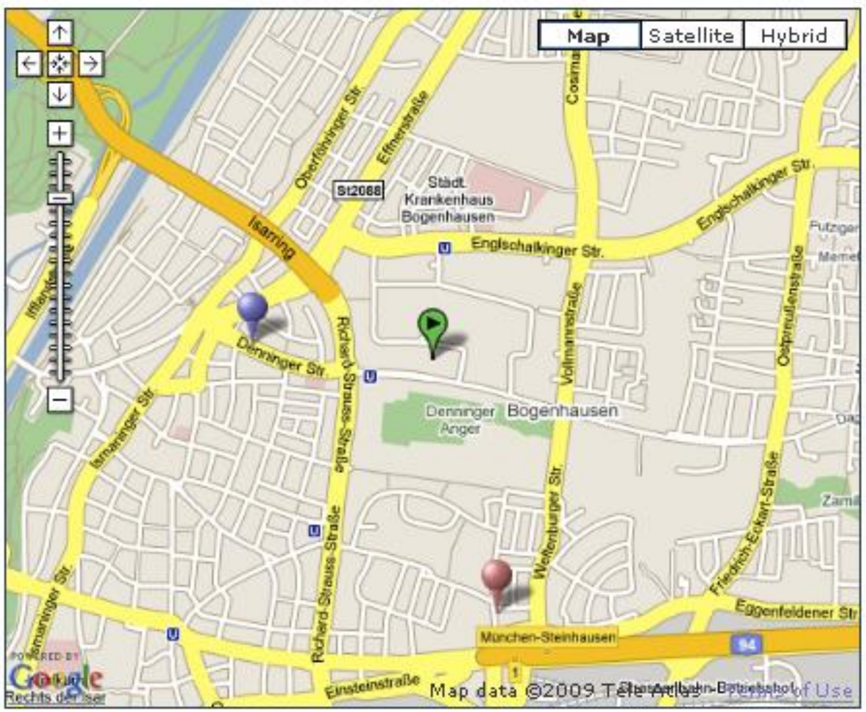
Sensoria

On Road Assistance

The best Garage
The best rental car station

Next step

start new service — continue



Current Location

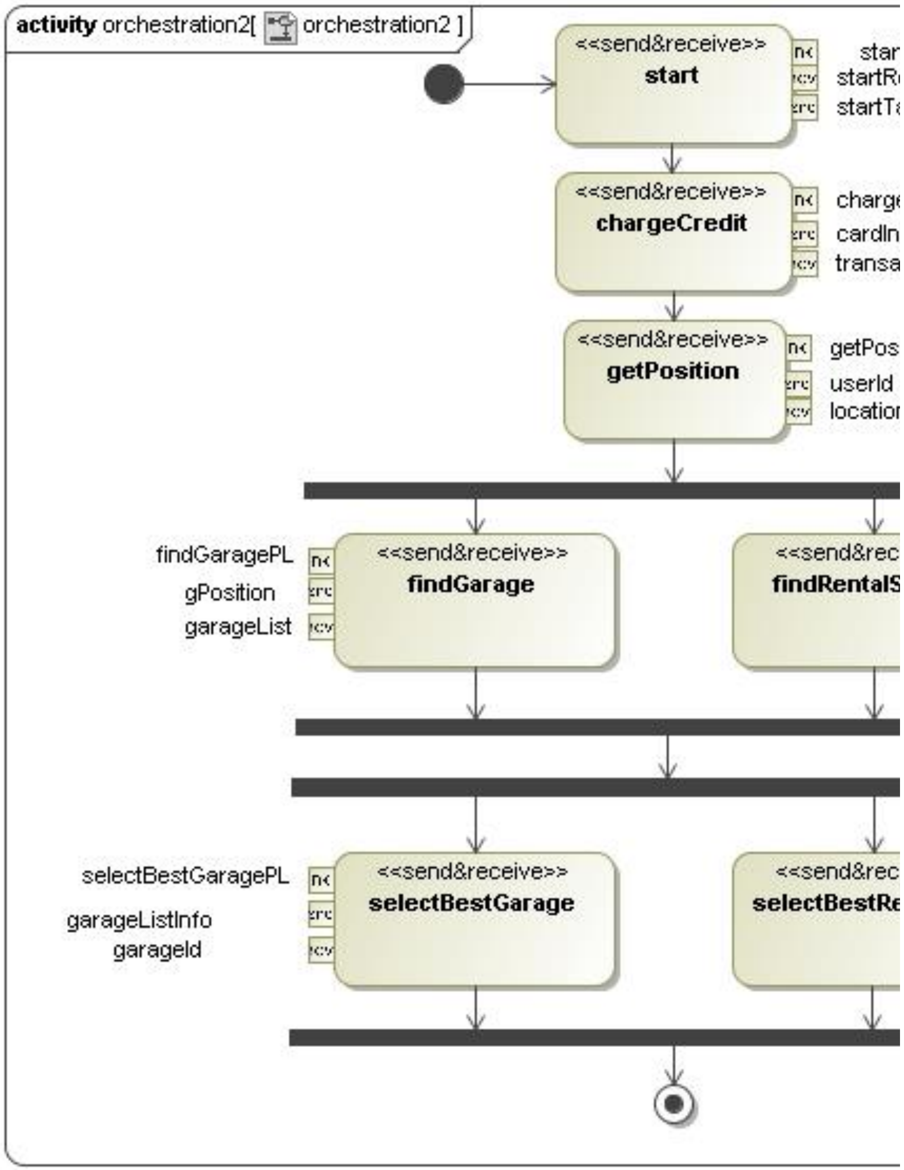
The best garage

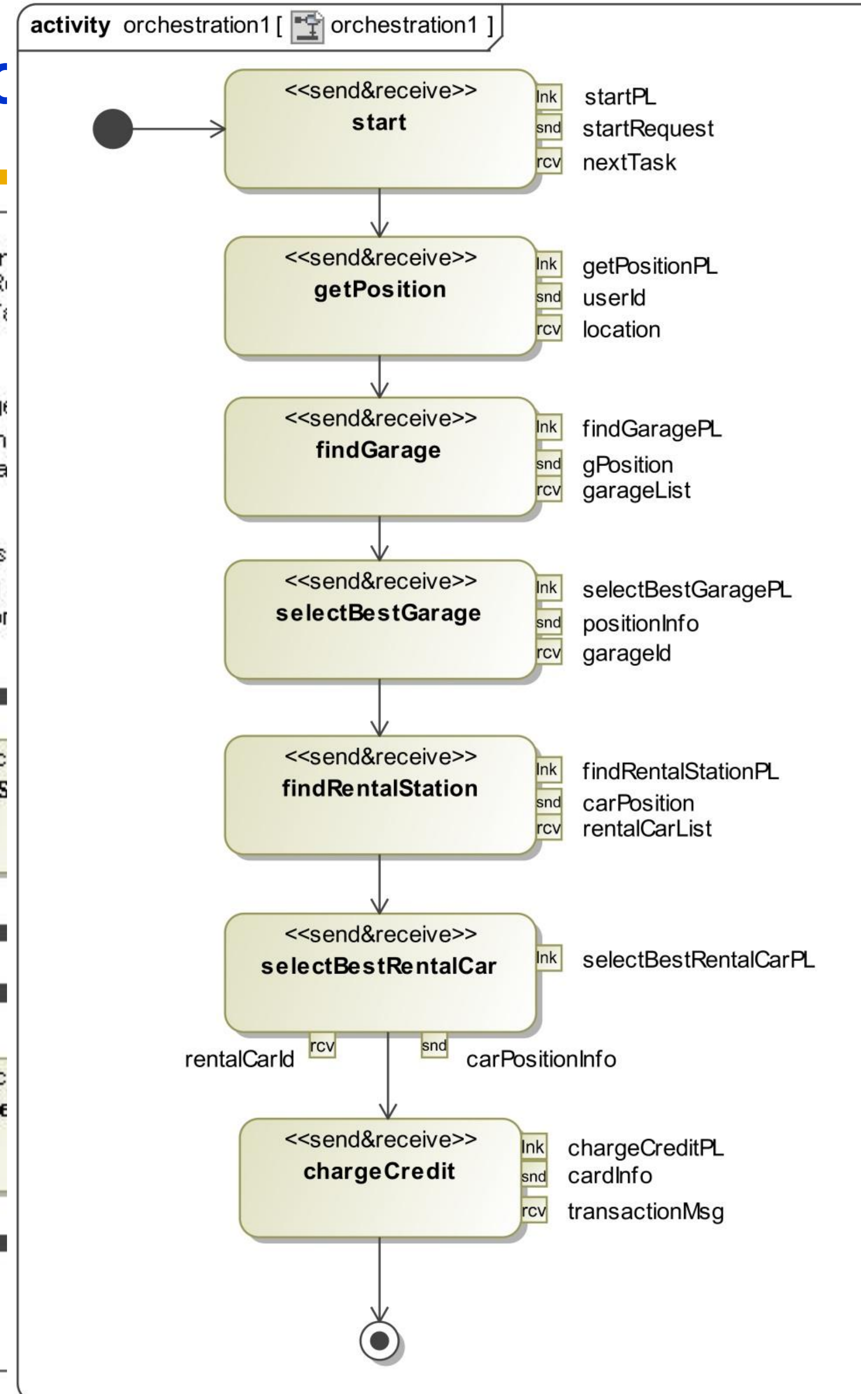
Garage Denninger — get route

The best rental car station

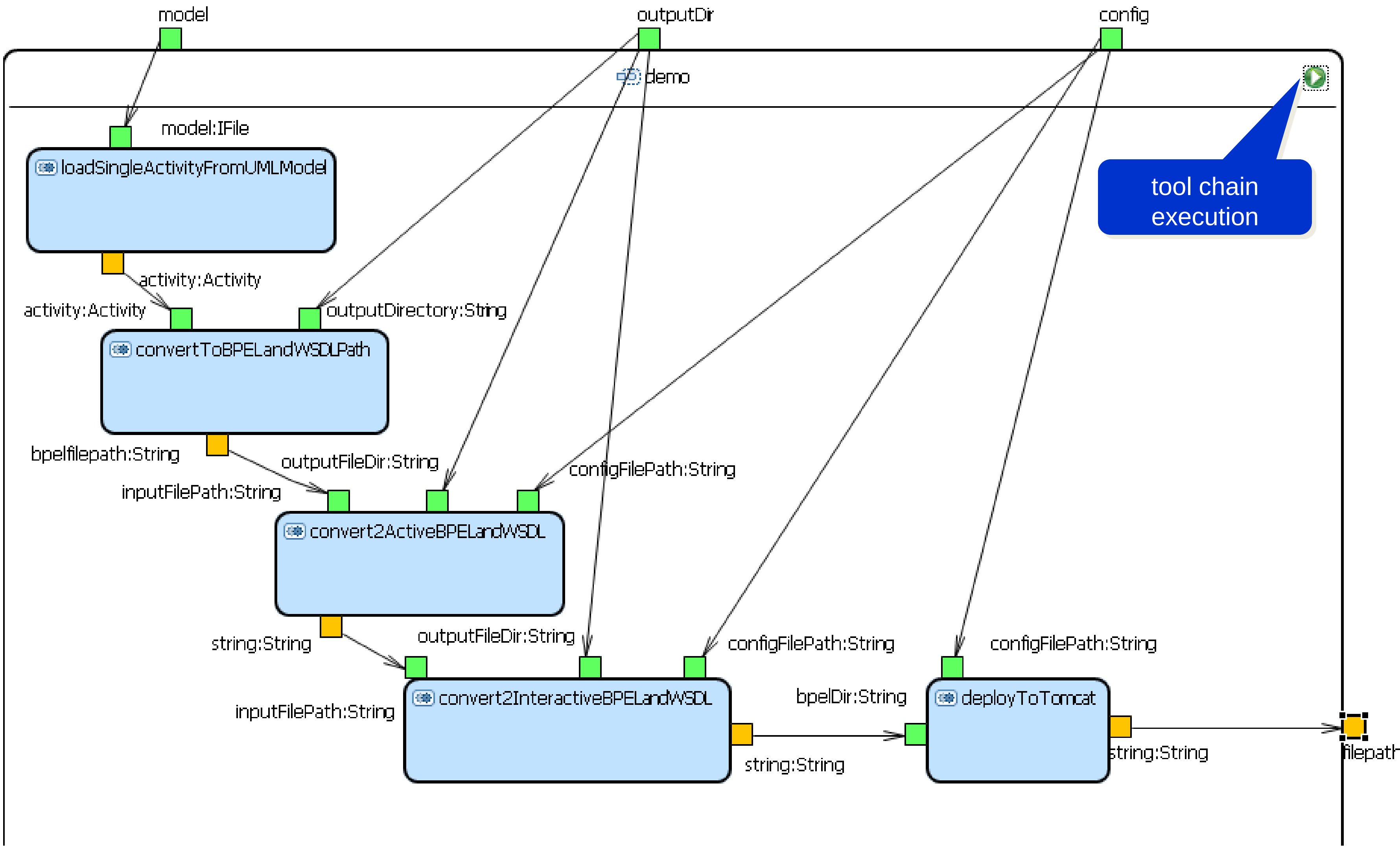
Car Rental Gotthelf — get route

7. Changing the orchestratio



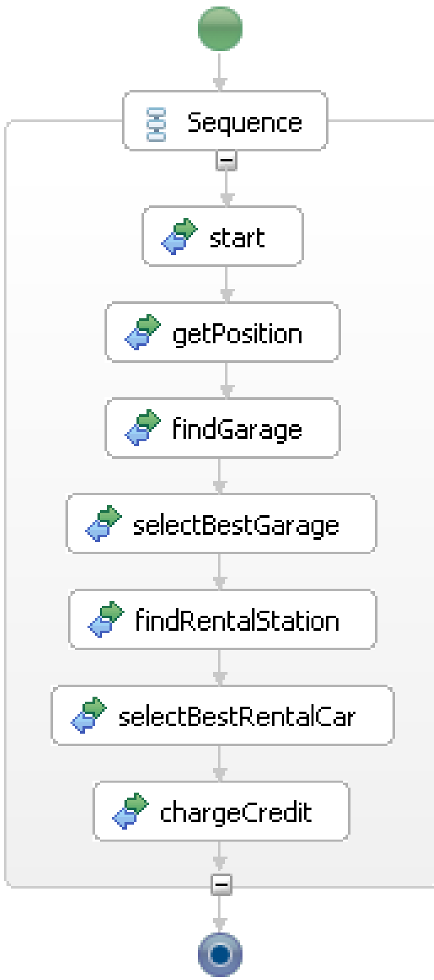


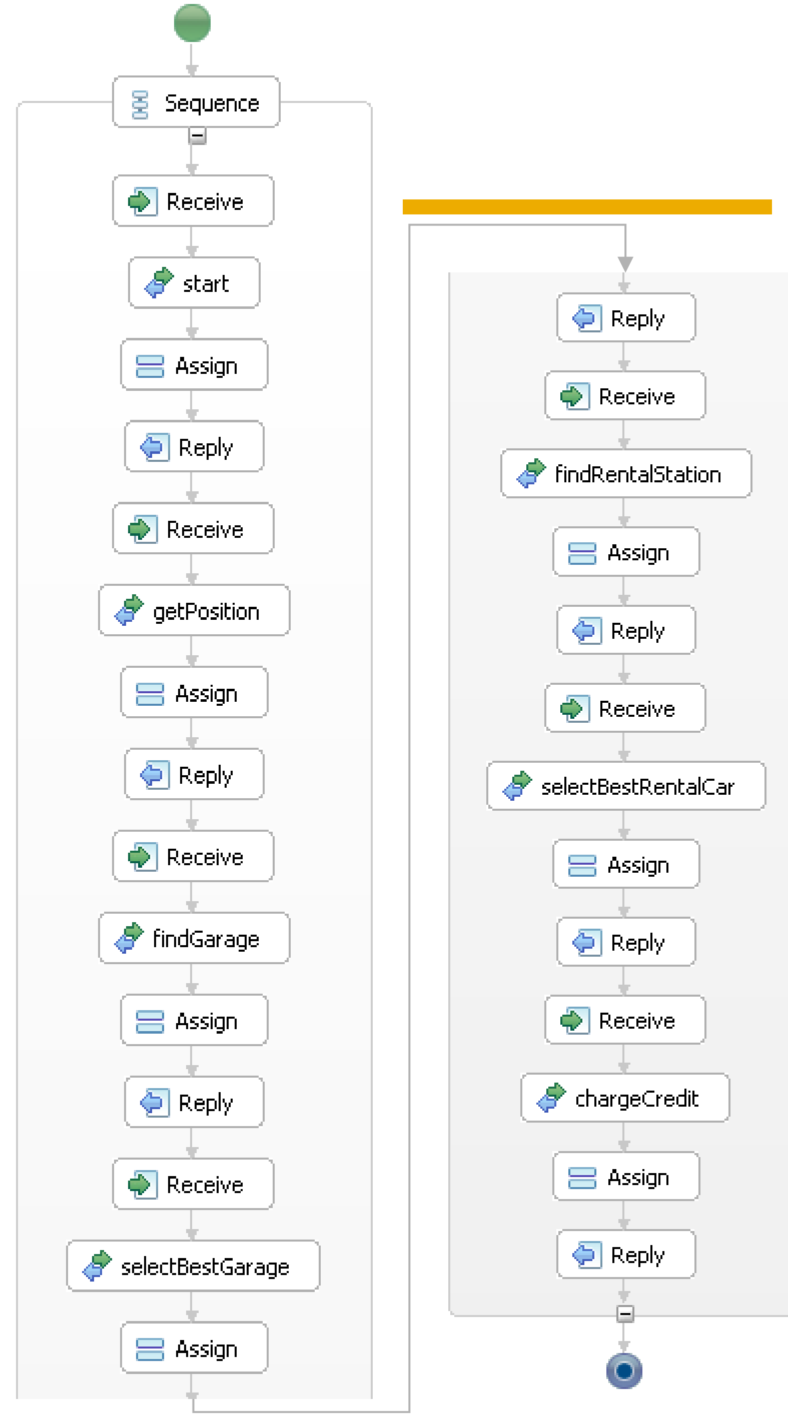
Back to the tool chain (step 3)



Looking at transformation re

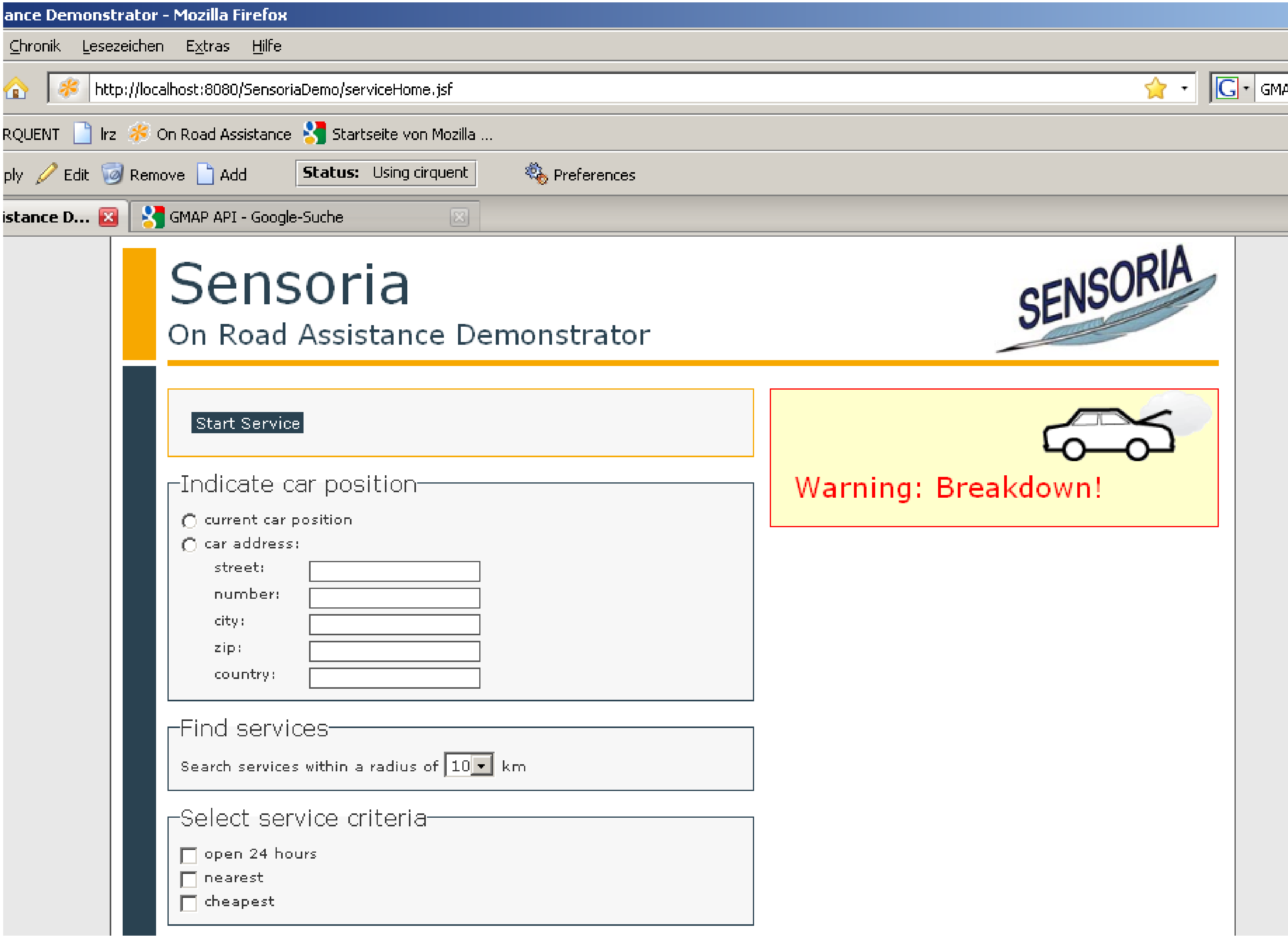
BPEL models





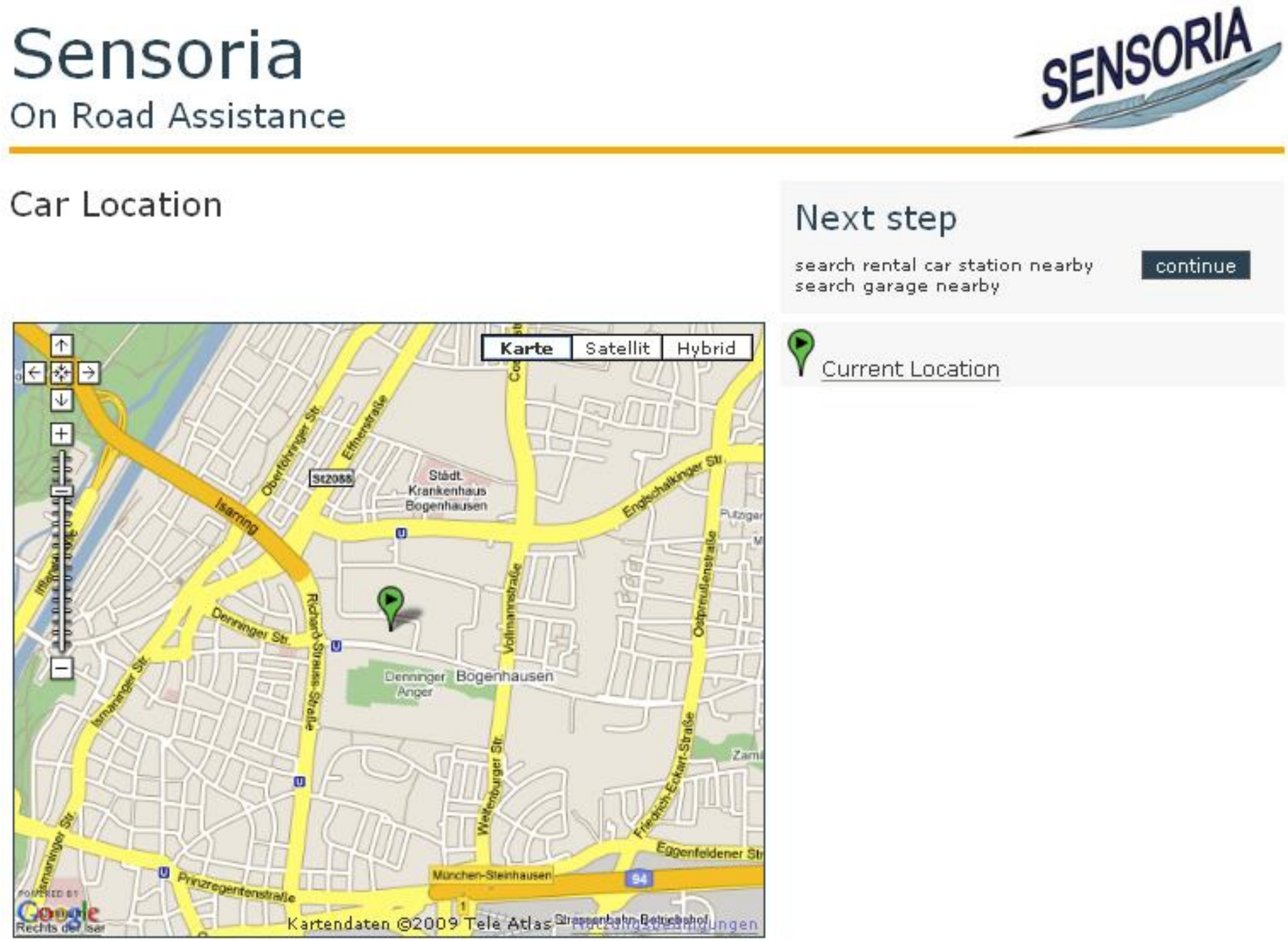
8. Running the deployed application again

Home Page - Setting of Preferences



8. Running the deployed application again

Car position



Conclusions

- Service Engineering Approach
  - modelling of SOSs
  - metamodels and UML profiles for SOC
  - transformations to analysis models
  - formal analysis of models
  - annotations of models
  - automatic generation of SOAs
  - pattern language
  - MDD4SOA@work

Bottom line: Ideas to take home

- Relevance of domain specific modelling language
  - UML profile
  - must be simple, few constructs
- Automated development approach
  - model-based and semantics driven
  - early qualitative and quantitative analysis based on formal techniques
  - model-driven (transformations)
  - pattern-based
- Importance of flexible tool support
  - easy (graphically) integration of diverse tools

References

- OMG, www.omg.org
- SENSORIA project, www.sensoria-ist.eu
- SHAPE project (SoaML), www.shape-project.eu
- SoaML, http://www.omg.org/spec/SoaML/