Software Engineering and Service-Oriented Systems

Martin Wirsing
LMU München

in co-operation with Francesco Tiezzi, and
the SENSORIA, REFLECT, and ASCENS teams
Ten Years Ago: 2004

- **IT-Hits**
  - New Browser: Mozilla Firefox 1.0
  - Siemens mobile campaign C70
  - Chip Magazin Innovation of the year: Sony Robot QRIO

- **Software Trends**
  - Embedded Systems
  - (Web) Services
  - Model-based Software Engineering

M. Wirsing

[blog.mozilla.org; www.chip.de; web-japan.org/kidsweb; www.w3.org]
Today: 2014

- **IT-Hits**
  - 2007 iPhone, 2008 iPad
  - 2014 Wearables
    - Apple Watch, Angel Sensor bracelet, …

- **Software Trends**
  - Virtualization (Cloud), “big data”
  - Personalized and physiological computing
  - Autonomic systems
  - Agile development

---

M. Wirsing

[www.apple.com; jewishbusinessnews.com/2013/10/21]
This Course

- From engineering service-oriented computing systems to
- engineering adaptive systems
  - Personalized physiological systems
  - Autonomous ensembles

Contents

- Engineering Service-Oriented Systems
- Engineering Physiological Computing Systems
- Engineering Autonomous Ensembles
Part 1: Engineering Service-Oriented systems

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
SENSORIA Project Overview and Results

- 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)
- Università di Trento
- University of Leicester
- Warsaw University
- Technical University of Denmark at Lingby
- Università di Pisa
- Università di Firenze
- Università 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
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)

- Issues in 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.
Modelling front-end

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

- **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.

- **Qualitative Analysis**
  - Venus verifier for properties of UML4SOA

- **Quantitative Analysis**
  - SRMC - SENSORIA Reference Markovian Calculus
Automated model transformations

Formal representations are generated by automated model transformations from engineering models, using the Service Development Environment SDE.
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
2. MDD4S0A – Model-Driven Development of Service-Oriented Systems

Martin Wirsing
LMU München

Nora Koch
LMU München and Cirquent GbmH

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


- 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

---

Transformation Rules

Generated Code

Code Generation Templates

Manually Written Code
SENSORIA Model-driven development

- Feedback
- Formal Methods
- Transformation
- Code Generation
- Modelling
- Reengineering
- Runtime
- Legacy Systems

Requirements
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

[Diagram of SLA_GM with interactions]

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

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

⇒ UML4SOA

⇒ 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 and
  - for reconfiguration, policies, requirements

- **MARTE profile** (OMG standard)
  - for performance and schedulability 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
SOA models in the MDA context

- **Computation Independent Model (CIM)**
  - **Business Model**
    - Enterprise Services
      - Roles, Collaborations, Dependencies, Workflows

- **Platform Independent Model (PIM)**
  - **Design Model**
    - Services
      - Components, Interfaces, Messages, Data

- **Platform Specific Model (PSM)**
  - **Technical Specification**
    - Technical Services
      - WSDL, BPEL, XML Schema, Java, Jolie

*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

- 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.
Identifying parties involved in SOAs

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.
A service contract is the specification of the agreement between providers and consumers of a service. It is modelled as a UML collaboration. A dependency represents the binding of the service contract to the provider or the consumer of the service.

- 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

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

Business Model

Enterprise Services
Roles, Collaborations, Dependencies, Workflows

Design Model

Services
Components, Interfaces, Messages, Data

Technical Specification

Technical Services
WSDL, BPEL, XML Schema, Java, Jolie

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

- 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.
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 services

- Service orchestration is the process of combining existing services to form a new service to be used like any other service.

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

M. Wirsing
Message passing
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

Pins containing interaction information
- **lnk**: partner
- **snd, rcv**: data to be sent or received

Use of variable after declaration
**Implicit** declaration of variable in a **snd** pin.
Data handling

**SoaML/UML4SOA**

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

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

---

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

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

- 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.
A `compensateAll` triggers all active compensation handlers in the reverse order.
## SOA model elements and diagram types

<table>
<thead>
<tr>
<th></th>
<th><strong>Business model</strong></th>
<th><strong>Design model</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Structural aspects</strong></td>
<td>capabilities, participants, service contract, service architecture, participant architecture</td>
<td>service point, request point, service interface, message type</td>
</tr>
<tr>
<td><strong>Behavioural aspects</strong></td>
<td>scope</td>
<td>send, receive, send&amp;receive, reply, raise, lnk, snd, rcv, compensate, compensateAll, compensation, exception, event, data</td>
</tr>
<tr>
<td><strong>Diagram type</strong></td>
<td>class diagram, composite structure diagram, activity diagram</td>
<td>class diagram, composite structure diagram, activity diagram, sequence diagram, state machine</td>
</tr>
</tbody>
</table>

+ *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

Template for a service level agreement (SLA)
Modelling a concrete configuration

Concrete SLA

- **Portal** (requester)
- **CreditRequest** (provider)

- **CreationValidationContract**
  - **ReliableMessagingCharacteristics**
    - **MsgSemantics**:
      - filterDuplicates = true
      - maxNumberOfRetrans = 3
      - needsAck = true
  - **Authentication**:
    - authToken = "username"
  - **Encryption**:
    - encryptBody = true
    - encryptAlgorithm = "default"
    - encryptHeader = false
    - encryptSignature = false
  - **DigitalSignature**:
    - signAlgorithm = "default"
    - signBody = true
    - signHeader = false
Example: UML4SOA metamodel
Conservative extension of the UML
SOA models in the MDA context

Computation Independent Model (CIM)  
**Business Model**

*Enterprise Services*
*Roles, Collaborations, Dependencies, Workflows*

Platform Independent Model (PIM)  
**Design Model**

*Services*
*Components, Interfaces, Messages, Data*

Platform Specific Model (PSM)  
**Technical Specification**

*Technical Services*
*WSDL, BPEL, XML Schema, Java, Jolie*

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

```java
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:
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, Hölz 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
    
    \[(X \rightarrow D) \rightarrow S\]
  - 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**  
\[ D = \{ n \in \mathbb{N} : 0 \leq 10000 \} \]

- **Semiring**  
\[ \text{FuzzyRing} = \langle R^+, \text{max}, \text{min}, 0, 1 \rangle \]
Soft Constraints and Preferences for Services

Soft constraint system for choosing the „best“ offer

- **Variables**  
  garage-cost, garage-duration, ...

- **Domain**  
  \( D = \{ n \in \mathbb{N} : 0 \leq 10000 \} \)

- **Semiring**  
  \( \text{FuzzyRing} = \langle \mathbb{R}^+, \text{max}, \text{min}, 0, 1 \rangle \)

- **Constraints and preferences**
  - Repair as soon as possible, in less than 48 hours
    \[ \text{fastRepair} : [\text{garage-duration} \mid n \mapsto [48/n]] \]
  - Private repair as cheap as possible, 1000 Euro and more almost unacceptable
    \[ \text{cheapRepair} : \text{in context} \neg \text{work-related}? \]
    \[ \text{assert} [\text{garage-cost} \mid n \mapsto [1000/n]] \text{ end} \]
  - Preference: Prefer fast repair to cheap repair
    \[ \text{fastRepair} > \text{cheapRepair} \]
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)**
  
  $\text{OnRoadAssistant} \parallel (\text{LocationSvc} \parallel \text{FindGrgeSvc} \parallel \text{FindRentalCarSvc} \parallel \text{CChargeSvc} \parallel \text{SelectGrgeSvc} \parallel \text{SelectRentalCarSvc})$

- **Determining the current location of the car and finding nearby services:**
  
  $\text{OnRoadAssistant} = (\text{start}, r_0) .
  (\text{chargeCredit}, \text{infty}) . (\text{getPosition}, \text{infty}).
  ((\text{findGarage}, \text{infty}) \parallel (\text{findRentCarStation}, \text{infty})).
  \text{OnRoadAssistant1}$

  $\text{LocationSvc} = (\text{getPosition}, r_2). \text{LocationSvc}$ ...

- **Selecting garage and rental car**
  
  $\text{OnRoadAssistant1} = ((\text{selectBestGarage},
  (\text{selectBestRentalCar}, \text{infty})). \text{OnRoadAssistant1})$

  $\text{SelectGrgeSvc} = (\text{selectBestGarage}, r_5). \text{selectGrgeSvc}$

  Passive waiting, not determining the rate

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

  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:**
  \[ 5 \times 5 \times 5 \times 5 \times 5 = \text{experiments with ipc/Hydra Tool [U. Edinburgh]} \]
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 \( r5 \)) is more important than trying to transmit location data faster (governed by rate \( r2 \)).
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 BPEL/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

- **outputDir**
- **model**
- **config**
Looking at transformation results: BPEL models
6. Running the deployed application

Home Page - Setting of Preferences

Sensoria
On Road Assistance Demonstrator

Start Service

Indicate car position:
- current car position
- car address:
  - street:
  - number:
  - city:
  - zip:
  - country:

Find services:
Search services within a radius of 10 km

Select service criteria:
- open 24 hours
- nearest
- cheapest

Warning: Breakdown!
6. Running the deployed application

Credit card charge

Sensoria
On Road Assistance

Payment Service

Please enter your credit card information: test
Name
Credit Card: MasterCard
Valid Until
Card Number
Security Number

submit
6. Running the deployed application

Car position

Sensoria
On Road Assistance

Car Location

Next step
search rental car station nearby
search garage nearby

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

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
7. Changing the orchestration model
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

Sensoria
On Road Assistance

Car Location

Next step
- search rental car station nearby
- search garage nearby

Current Location
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