Aim of the tutorial

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

I. Setting the scene
   - the context – SENSORIA
   - what we mean by “service-oriented systems”
   - what we mean by “model-driven development”

II. Engineering of service-oriented systems
   - development process
   - modelling
   - metamodel and model transformations
   - tool support
   - model-driven development @ work
   - pattern language

I. Setting the Scene
Context

Software Engineering for Service-Oriented Overlay Computers

- EU project
- 19 partners
- 2005 - 2010
- more than 500 publications
- 2 spin offs

… more details

- 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
Project overview

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

... further details

www-sensoria-ist.eu

industrial and academic forum
What we mean by “service-oriented systems”

- **Service**
  - autonomous, platform-independent computational entity that can be described, published, categorised, discovered
  - services can be consumed without having to care about their maintenance, destruction, etc. (difference to components)
    - 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.

... more terminology

- **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
    - “distributed computing” is such another paradigm

- **Service-Oriented Architecture (SOA)**
  - an architectural style to realize SOC
    - “client/server” is an architectural style for realizing distributed computing
Stakeholders/Parties in service-oriented architectures

- 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

What is a “model”

- A *description* of (part of) a system written in a *well-defined* language
  (Equivalent to *specification*) [Kleppe, 2003]
- A *description or specification* of the system and its environment for some certain *purpose*. A model is often presented as a combination of drawings and text. [MDA Guide, 2003]
Examples

- City models
  - underground and bus maps, commuting models,...
- Building/house maps
  - floor plans, electric wiring, water and central heating pipes, etc.
- Scientific models
  - mathematical models
  - statistical models
  - simulation models
- Software system models
  - requirements (use cases)
  - structure (class diagrams)
  - behavioural models
  - choreography models
  - load balancing and deployment models,...

MDA proposed “everything is a model”
- a process is a model
- a platform is a model
- a transformation is a model
- a metamodel is a model
- a system is a model
- a program is a model
- a measure is a model
- a test is a model
- a pattern is a model
- ...

Characteristics of models

- Abstract
  - emphasize important aspects, hide irrelevant ones
- Understandable
  - expressed in a form readily understood by users
- Accurate
  - faithfully represent the modelled system
- Predictive
  - can be used to derive correct conclusions about the system
- Inexpensive
  - cheaper to construct and study than the system
Usefulness of models

- Specify the system
  - structure, behaviour, ...
  - separate concepts at different conceptual levels
  - communicate with stakeholders
- Understand the system
  - if existing (legacy applications)
- Validate the system
  - detect errors and omissions in design ASAP
  - mistakes are cheaper at this stage
  - prototype the system (execution of the model)
  - formal analysis of system properties
- Drive implementation
  - code skeleton and templates
  - complete programs (if possible)

What is meant by “model-driven development”

- Model-Driven Development/Engineering (MDD™/MDE)*
  - refers to a range of engineering approaches that are based on the use of software models as a primary form of expression
  - has a focus on architecture and corresponding automation
  - objective is to generate code from the models
- Model-Based Development
  - instead expresses that models are mainly used for communication and documentation
- Model-Driven Architecture (MDA™)*
  - is the best known MDE initiative

* Note that MDA, MDD are trademarks of the OMG; MDE is not
MDA terminology

- Computational Independent Model (CIM)
  - describes the business context and business requirements
  - focuses on the environment of the system

- Platform Independent Model (PIM)
  - specifies structure and functionality of the software system
    independent of software technology platforms
  - suitable for use with a number of different platforms

- Platform Specific Model (PSM)
  - describes the realization of the software systems with
    respect to the chosen software technology platforms

MDA in a nutshell

- MDA supports the idea of
  - designing software systems using model(s) in the development
    - CIM, PIM, PSM
  - transforming CIMs to PIMs, PIMs to PIMs and PIMs to PSMs
    - based on model transformation technologies
  - models are first class entities

- MDA promotes to build different views (models) of a system following a separation of concerns

- MDA/MDE is changing the software development paradigm from code-centric to model-centric

*More about MDD,MDE, MDA, ... later*
II. Engineering of Service-Oriented Systems

Motivation

- Service-oriented architectures (SOAs)
  - promise to organize and understand organizations, communities and systems maximizing agility, scalability and interoperability
  - built by IT industry in an ad-hoc and undisciplined way

- Challenges for service-oriented computing (SOC)
  - specification of correct behaviour of SOAs
  - automated composition of services (orchestration)
  - long running transactions
  - performance, security and safety
  - deployment and re-engineering
**MDE in SENSORIA**

- SENSORIA approach to model-driven service engineering
  - from business models to implementations
  - via model transformations
- Formal analysis
  - functional service verification
  - type correctness
  - sensitivity analysis
  - scalability analysis
- Flexible service development support
  - service development patterns
  - development environment

**... 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 model-driven development process

Analysis
Verifying properties of SOA models

Improvement
Preparing results for improving models

Transformation
Translating models to formal languages

Design
Modelling SOA applications

Business Modelling
SOA Architecture
e.g. requirements

Code Generation
Creating executable code, e.g. BEPL/WSDL

SENSORIA Development Environment

Project results

- Languages
- Techniques
- Methods
- Tools

to support this development process of service-oriented systems
... concrete results

- Service ontology
- Modelling languages
  - UML4SOA, SRML, SITowla
- Process calculi
  - SCC, SOCK, Stock, COWS, ...
- 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 (continued)

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

- **Analysis**: Verifying properties of SOA models
- **Transformation**: Translating models to formal languages
- **Design**: Modelling SOA applications
- **Code Generation**: Creating executable code, e.g. BEPL/WSDL
- **Business Modelling**: SOA Architecture, e.g. requirements
- **Improvement**: Preparing results for improving models

**SENSORIA Development Environment**

Modelling languages

- **Objective**: 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
  - **Option 3**: Define a UML2 profile
    - using the extension mechanism that allows to customize the UML for specific domains and platforms
    - defining stereotypes, 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

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 standardization process beta1 version)
  - 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
Service Oriented Architecture Modeling Language

- 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 Innsbruck, University of Augsburg, University of Athens
  - SHAPE project (FP7) is the main contributor
- Results
  - Merge of approaches, June 2008
  - 1st revised submission, August 2008
  - 2nd revised submission, November 2008
- Meetings SoaML and UML4SOA groups
  - next, Sept. 2009

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

- **Business Model**
  - Computation Independent Model (CIM)
    - 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.
Process as orchestration of services

- UML activity diagram selected for the representation of orchestration of services

A scope is used to group service specific actions and scopes. It may have associated event, exception and compensation handlers. A scope can be represented as a UML StructuredActivityNode or an Activity.

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

Modelling service contracts

- 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 participant can play different roles.

- 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.
A services 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.

Participant architecture

- It is important not to over-specify any of the parties, i.e. usually it is not required to specify the internal structure of a participant allowing each party maximum freedom in how they achieve their goals.
- However, it is possible to provide a high-level services architecture of a participant.
- Defines how a set of internal and external participants use services to implement the responsibilities of the participant.
SOA models in the MDA context

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

---

Refining specification of participants with ports

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

![SoaML Diagram]

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

- 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 service interface**
  - implements ("provides") provider interfaces (represented as realisation)
  - "requires" consumer interfaces (represented as a «use» dependency)

---

Service point and request point
Reverse interfaces

- A **service point** provides the provider interface and requires the consumer interface.
- An **request point** requires the provider interface and provides the consumer interface (reverse of service point).

- A **service interface** implements ("provides") provider interfaces (represented as realisation)
- A **service interface** "requires" consumer interfaces (represented as a «use» dependency)
Service point and request point
Reverse Interfaces

SoaML

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

Change in UML Metamodel
Required by SoaML

- Port is modified to indicate the direction of a Port, whether
  - the Port is providing the operations available through a Port or
  - the Port is consuming them
- Current situation in the UML
  - conjugate types must be created explicitly
Interface behaviour

- **UML4SOA**
  - propose protocol state machine
- **SoaML**
  - propose activity diagrams or sequence diagrams

![Diagram of a UML state machine]

Service channel

- Communication path between service points and request points within an architecture that
  - connects consumers and providers
  - defines the coupling in the system
  - extends UML connector

- Connection possible if request and service point are compatible:
  - both have the same port type (interface or service interface)
  - type of the service point is a specialisation or realisation of the type of the requested point
  - both have compatible needs and capabilities, i.e. the service must provide an operation for every operation used through the request and reverse
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

Message passing among requester and provider

Synonymous and asynchronous service invocation

- Service interaction `send` sends a message. Does not block.
- Service interaction `receive` blocks until message is received.
- A `raise` action causes normal execution flow to stop and invokes associated exception handlers.
- `Reply` is used for the reception of a message decoupled of the sending process.

Service interactions `send&receive, receive&send` denote a sequential order of these actions.
**Detailing service invocation**

**Partner services and data handling**

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

- **Implicit declaration of variable in a rev pin.**

- **Use of variable after declaration**

- **Variables belong to the scope they are declared in**
  - Use of "::" for referring of variables of parent scopes

**Accept activity is part of the scope Decision**

---

**Data handling**

- **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 scope modelling the compensation handler will be triggered by a compensate or compensateAll.

Compensation

A compensateAll triggers all active compensation handlers.
SOA model elements and diagram types

<table>
<thead>
<tr>
<th>Business model</th>
<th>Design model</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Structural aspects</strong></td>
<td>capabilities</td>
</tr>
<tr>
<td></td>
<td>participants</td>
</tr>
<tr>
<td></td>
<td>service contract</td>
</tr>
<tr>
<td></td>
<td>service architecture</td>
</tr>
<tr>
<td></td>
<td>participant architecture</td>
</tr>
<tr>
<td><strong>Behavioural aspects</strong></td>
<td>scope</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Diagram type</strong></td>
<td>class diagram</td>
</tr>
<tr>
<td></td>
<td>composite structure diagram</td>
</tr>
<tr>
<td></td>
<td>activity diagram</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></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)

Concrete configuration

Concrete SLA
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...

What is meant by “metamodel”

- A model of a modelling language [Seidewitz, 2003]
  - That is, a metamodel makes statements about what can be expressed in the valid models of a certain modelling language.

- A model that defines the language for expressing a model [MOF, 2000]

- A meta-metamodel is a model that defines the language for expressing a metamodel, e.g. Meta Object Facility (OMG). The relationship between a meta-metamodel and a metamodel is analogous to the relationship between a metamodel and a model.
MDA principles

- Models are specified using a modelling language (M₁)
- A modelling language is described by a metamodel (M₂)
- Metamodels belong to a library of domain specific languages (DSLs)
- Metametamodel: there is a unique language for describing these metamodels (M³), i.e. the Meta Object Facility (MOF)

Four-layers metamodel hierarchy

- System represented by the real world
- Model represented by the modelling world
- Metamodel
- Metametamodel

Conforms to:
- M³ to M₂
- M₂ to M₁
- M₁ to M₀

Metamodels:
- M₂: SPEM, UML, CWM
- M₁: a UML model, another UML model
- M₀: a particular use of a UML model, another use of a UML model

© Nora Koch
Four-layers metamodel hierarchy (example)

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. Ink)
- Use of extension mechanisms of the UML
  - stereotypes
  - tagged values
  - constraints
- Risks
  - too many stereotypes
  - selection of inadequate UML metaclass
  - decorative and redefined stereotypes (→)

## Types of UML extensions

- Decorative
  - vary only the concrete syntax (visual presentation)
  - few value
- Descriptive
  - extend the syntax of a language such that additional information can be expressed
  - limited power as purely syntactical
- Restrictive
  - descriptive and impose semantic restrictions
  - has the capability to define a meta language on top of the base language
- Redefined
  - modify the core semantics of the language elements
  - no need of a base language

Gliz et al., UML 1999
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 ( #3 is false)
  2. identification of appropriate *UML metaclasses*
  3. stereotypes and metamodel elements related by an “extension” (multiple metaclasses)
  4. define *semantics* of new elements

Several profiles for SOAs in SENSORIA

Profiles for modelling different aspects of SOSs: structural aspects (SoaML), behavioural aspects (Orchestration), non-functional properties (NFP), reconfiguration (Modes), policies (SiPowla), requirements (SRML), and performance (MARTE).
UML4SOA metamodel: Orchestration Package
Conservative extension of the UML

SoaML metamodel
Specification of „new“ elements

- Service Interface (excerpt)
  - Description
    - defines the interface to a Service Point or Request Point and is the type of a role in a service contract...
  - Extended Metaclass
    - Class
  - Attributes
    - no new attributes
  - Associations
    - no new associations
  - Constraints
    - A Service Interface must not define the methods for any its provided operations or signals….
  - Semantics
    - A Service Interface defines a semantic interface to a Service or Request. That is, it defines both the structural and behavioural semantics of the service necessary for consumers to determine if a service typed by a Service Interface meets their needs, and for consumers and providers to determine what to do to carry out the service…

- Notation
- Examples
- Additions to UML2

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!")()
}
```

Integration with other technologies
- Jolie can be used for interacting with existing web services, own services and integrating legacy systems and current technologies within a SOA

- Open-source project
- Java Orchestration Language Interpreter Engine
- Spin-Off of the University of Bologna
Verification in model-driven service engineering

Business Modelling
SOA Architecture
e.g. requirements

Design
Modelling SOA applications

Analysis
Verifying properties of SOA models

Improvement
Preparing results for improving models

Transformation
Translating models to formal languages

Code Generation
Creating executable code, e.g. BEPL/WSDL

SENSORIA Development Environment

Quantitative and qualitative analysis methods

- Analysis using formal techniques
  - performance analysis
  - service level agreement analysis
  - security and behavioural analysis
- Methods and tools based on
  - stochastic simulation
  - model checking
  - logic
- Model-based analysis
  - in early phase of the development process
Performance analysis at model level

- Using formal techniques for SOA
  - prediction of service level agreement and performance
  - annotation of UML diagrams with rates of time consuming actions of the workflow (stereotypes of MARTE profile)
  - translation of the activity diagrams into stochastic process calculus PEPA
  - prediction with the tool SRMC (SENSORIA Reference Markovian Calculus)

PaStep: a basic sequential execution step on a host processor.

Model transformations

- Goal is automatic translation between source and target models
- Translation performed by a transformation engine that executes transformation rules
- Set of rules
  - seen as a model
  - based on a transformation metamodel
- MDA model transformations
  - CIM2PIM
  - PIM2PIM
  - PIM2PSM

Model transformation pattern (J. Bézivin, 2004)
MDD4SOA and VIATRA

- **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

- **VIATRA2 ((V)Visual Automated model TRAnsformations )**
  - general tool based on graph transformations and abstract state machines
  - used within the project for deployment transformations
  - Eclipse project

---

**Tool support**

<table>
<thead>
<tr>
<th>Tool</th>
<th>Area</th>
<th>Case Study</th>
<th>Integratio n in SDE</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDE</td>
<td>Integration</td>
<td>All</td>
<td>X</td>
</tr>
<tr>
<td>UML4SOA profile</td>
<td>Modelling</td>
<td>Finance, eUniversity,</td>
<td>X</td>
</tr>
<tr>
<td>MDD4SOA Transformer</td>
<td>Transformation</td>
<td>Finance, eUniversity,</td>
<td>X</td>
</tr>
<tr>
<td>VIATRA2</td>
<td>Transformation</td>
<td>Finance, eUniversity</td>
<td>X</td>
</tr>
<tr>
<td>LTSA/WS-Engine</td>
<td>Analysis</td>
<td>eUniversity</td>
<td>X</td>
</tr>
<tr>
<td>PEPA</td>
<td>Analysis</td>
<td>eUniversity</td>
<td>X</td>
</tr>
<tr>
<td>LySa</td>
<td>Analysis</td>
<td>Finance</td>
<td>X</td>
</tr>
<tr>
<td>UMC/CMC</td>
<td>Analysis</td>
<td>Automotive</td>
<td>X</td>
</tr>
<tr>
<td>DINO</td>
<td>Runtime</td>
<td>Finance, Automotive</td>
<td>X</td>
</tr>
<tr>
<td>SRML Editor</td>
<td>Modelling</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ADR2GRAPHS</td>
<td>Visualisation</td>
<td>Automotive</td>
<td>-</td>
</tr>
<tr>
<td>Jolie</td>
<td>Modelling</td>
<td>eUniversity</td>
<td>X</td>
</tr>
<tr>
<td>Reengineering Tool</td>
<td>Reengineering</td>
<td>Finance</td>
<td>-</td>
</tr>
</tbody>
</table>
**SDE: SENSORIA Development Environment**

- **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

---

**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
     - 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

- Diagram:
  - Build orchestration model
  - Transform model
  - Generate application
  - Deployment
  - Run application
  - Change model
  - OK
1. Building an orchestration model with UML4SOA

- Automotive Case Study: 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 driver's 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)
2. 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)
3. Executing tool chain

Looking at transformation results
BPEL model
4. Running the deployed application

Home Page - Setting of Preferences

Sensoria
On Road Assistance Demonstrator

Warning: Breakdown!

Sensoria
On Road Assistance

Payment Service

Please enter your credit card information, text

Name
Credit Card
Valid Until
Card Number
Security Number

submit
4. Running the deployed application

Car position

Sensoria
On Road Assistance

Garage and rental car services

Sensoria
On Road Assistance
4. Running the deployed application
Selection best services

Sensoria
On Road Assistance

The best Garage
The best rental car station

Next step
Start new service

5. Changing the orchestration
6. Back to the tool chain

Looking at transformation results: BPEL model
4. Running the deployed application again

Home Page - Setting of Preferences

4. Running the deployed application again

Car position
4. Running the deployed application again

- Different order of web pages
  - Credit card charge at the end
  - Only list of garages
  - etc.

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-in)
  - for solving a particular problem arising in the development process

- To ameliorate this problem we are developing a catalogue of patterns that
  - serves as an index to our results
  - illustrates, in a concise manner, the advantages and disadvantages of the individual techniques
Scalability analysis pattern

- Context
  - a large-scale service provider using replication to scale his service provision to support large population.
- Problem
  - understanding the impact of changes in number of servers or number of users subscribed to his service.
- Forces
  - being able to support large-scale use is an indicator of quality in planning.
  - heavy demand due to large user populations require service replication, but replication represents costs.
- Solution
  - develop a high-level model of the system and apply continuous-space analysis to the model to make predictions about the large-scale system.
- Related patterns
  - sensitivity analysis.
- Tools
  - PEPA Eclipse plug-in project.

Patterns catalogue

- Patterns defined so far …
  - Service modelling.
  - Service specification and analysis.
  - Functional service verification.
  - Sensitivity analysis.
  - **Scalability analysis**.
  - Declarative orchestration.
  - Model-driven development.

---

A pattern-based approach to augmenting service engineering with formal analysis, transformation and dynamism, Martin Wirsing et al., ISOLA 2008.
Conclusions

Service Engineering Approach
- modelling of SOSs
- metamodels and UML profiles for SOC
- transformations to analysis 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
  - model-driven (transformations)
  - pattern-based
- Importance of flexible tool support
  - easy (graphically) integration of diverse tools
Thank you for your time and attention!

Questions? Comments?

Nora Koch
kochn@p4t.ifi.lmu.de

Further information
www.sensoria-ist.eu