www.sensoria-ist.eu

# The SENSORIA Development Environment

## Goals, Benefits, Implementation and Usage

Philip Mayer
Programming and Software Engineering Research Group
Department of Informatics
LMU Munich
Germany

Information Society Technologies

# Outline



- Sensoria break-down: Formal methods & tools
- From a tooling perspective: How Sensoria views SOA development
- Introducing the SDE
  - SDE Aims
  - SDE Design & Implementation
- Tools in the SDE
- Using the SDE
- SDE Internals

- Sensoria stands for **Software Engineering for Service-Oriented Overlay Computers**
- Sensoria is an EU-sponsored research project spanning 3 universities, 2 research institutes and 4 companies from 7 European countries

The aim of Sensoria is the development of **new methodologies and tools** for the development of **Service-Oriented Software Systems**

# Sensoria Output

- Sensoria aims at providing researchers and practitioners with concepts and tools for
  - modeling,
  - checking, analyzing, and verifying,
  - transforming,
  - and deploying

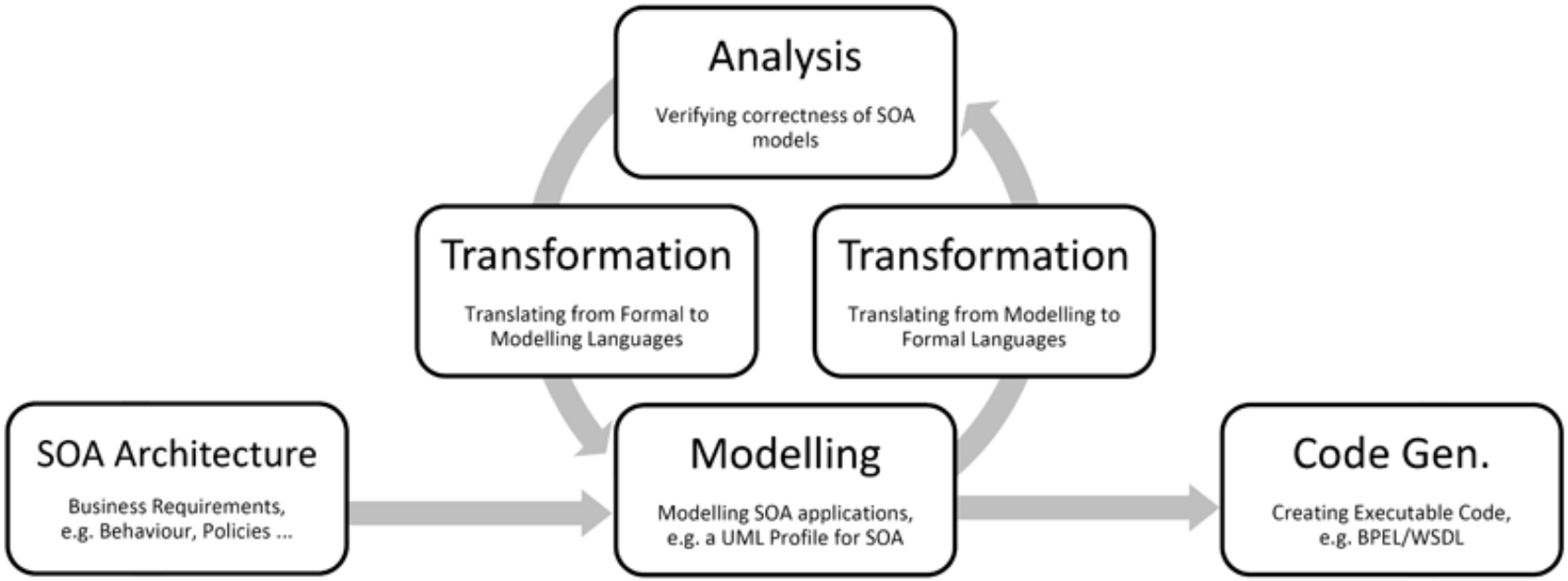
  ...service-oriented software systems

- Some of the results are **highly theoretical**, others result in **tools** (and some in **both**).
- In this talk, I will focus on a tooling perspective on Sensoria, i.e. the question

**How can we support, through tools based on formal methods, the development process of Service-Oriented Systems?**

# Supported Steps in the Development Process

The Sensoria project provides solutions (tools) for each of these areas:



# Tools, Tools, Tools

- There are quite a few tools developed within Sensoria, over a dozen are listed on the website
- Initially, they were developed independently and did not cooperate with one another
- However, there are several scenarios in which developers might want to use a combination of these tools
  - For example:
    a. Modelling in UML
    b. Checking the model using a formal language
    c. Transforming automatically in-between those two.
  - Three tools for one task!

- The question was how to bring all these tools together
- The answer is a **lightweight integrated development environment** to with the tools can be contributed.

## Requirements for an integration tool

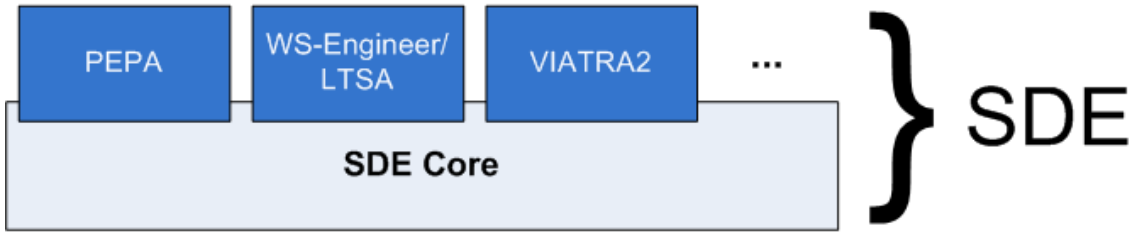


- An integration tool should
- ...for **users**: Enable them to use discover and use tools alone or in combination, with the ability to write orchestrations of tools to use as a new tool (idea of service orchestrations).
- ...for **developers**: Enable easy integration of a tool API, and straightforward publishing of tools (idea of yellow pages)
- We address these requirements with the **Sensoria Development Environment (SDE)**
- **Benefits** include
  - A means of providing SENSORIA tools from a central location
  - A homogeneous platform for using the SENSORIA tools
  - Ability to compose tools and run tool chains automatically

# The Sensoria Development Environment

- The Sensoria Development Environment is a **lightweight OSGi/Eclipse-based integration platform for developing SOA-based software**

- The SDE consists of the integration platform (**SDE Core**) and integrated tools:




- Tools are registered with the SDE Core
- The core also handles the registry (finding tools) and tool automation (through scripts or orchestrations)

# Key Features of the SDE/SDE Core

- **A SOA-based platform**
  - The SDE Core is based on a Service-Oriented Architecture itself
  - The tools hosted in the SDE Core are registered and handled as services
  - A service registry allows discovery of tools
- **A Composition Infrastructure**
  - Composition, or orchestration, of services is a key concept of SOAs
  - It allows composing existing services to form a new one
  - The SDE Core enables this process for tools to automate commonly used workflows
- **A Focus On Usability**
  - Many Sensoria tools are based on formal methods and languages
  - The idea is to expose as much functionality and as little low-level code as possible
  - The SDE architecture encourages use of automated model transformations to translate between high-level models and formal specifications

# Tools in the SDE

- A **tool**, integrated into the SDE, is an entity with a name, a description, etc, and a list of invokable functions (API)
- A **function** is the basic execution unit of a tool. It takes arbitrary (developer-defined) parameters and may return an arbitrary object as well.
  - A function has attached metadata which describes the parameters and return types in more detail.
- Furthermore, each tool may contain **options** which can be changed through the framework.

- Once a user has retrieved a tool from the SDE registry, invocation of functions is done directly.
- The SDE also contains a UI for talking directly to the tools (mainly for debugging reasons).

# Advanced features

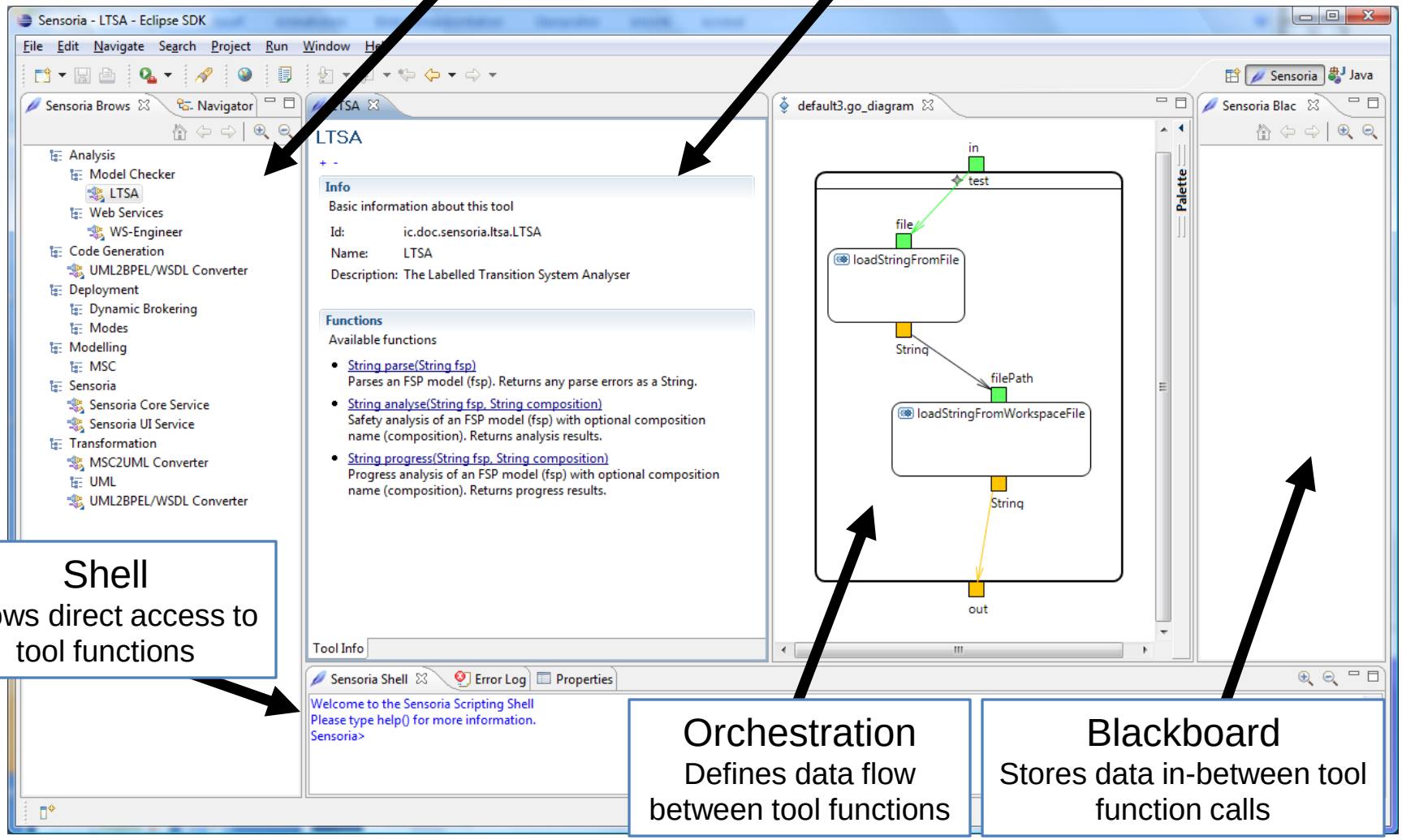
- A particularly useful feature of the SDE is tool **composition**, or **orchestration**:
  - All tools within the SDE provide a public interface
  - This interface can be used from orchestration languages (text-based or graphical) to create new tools combined from existing ones
- The SDE contains two default orchestration mechanisms
  - A JavaScript-based textual script editor + execution environment
  - A graphical (UML activity diagrams-like) orchestration editor with cross-translation to Java for execution.
- Another feature of the SDE is **remote invocation**
  - The SDE core is inherently distributed
  - I.e., each installation contains a networking core which can be linked with other through R-OSGI (Remote OSGi)
- This allows using tools installed on other machines, and transferring data between them.

# Using tools

www.sensoria-ist.eu

**Tool Browser**
Lists available tools, ordered by category

**Function Browser**
Lists functionality available in a tool




**Shell**
Allows direct access to tool functions

**Orchestration**
Defines data flow between tool functions

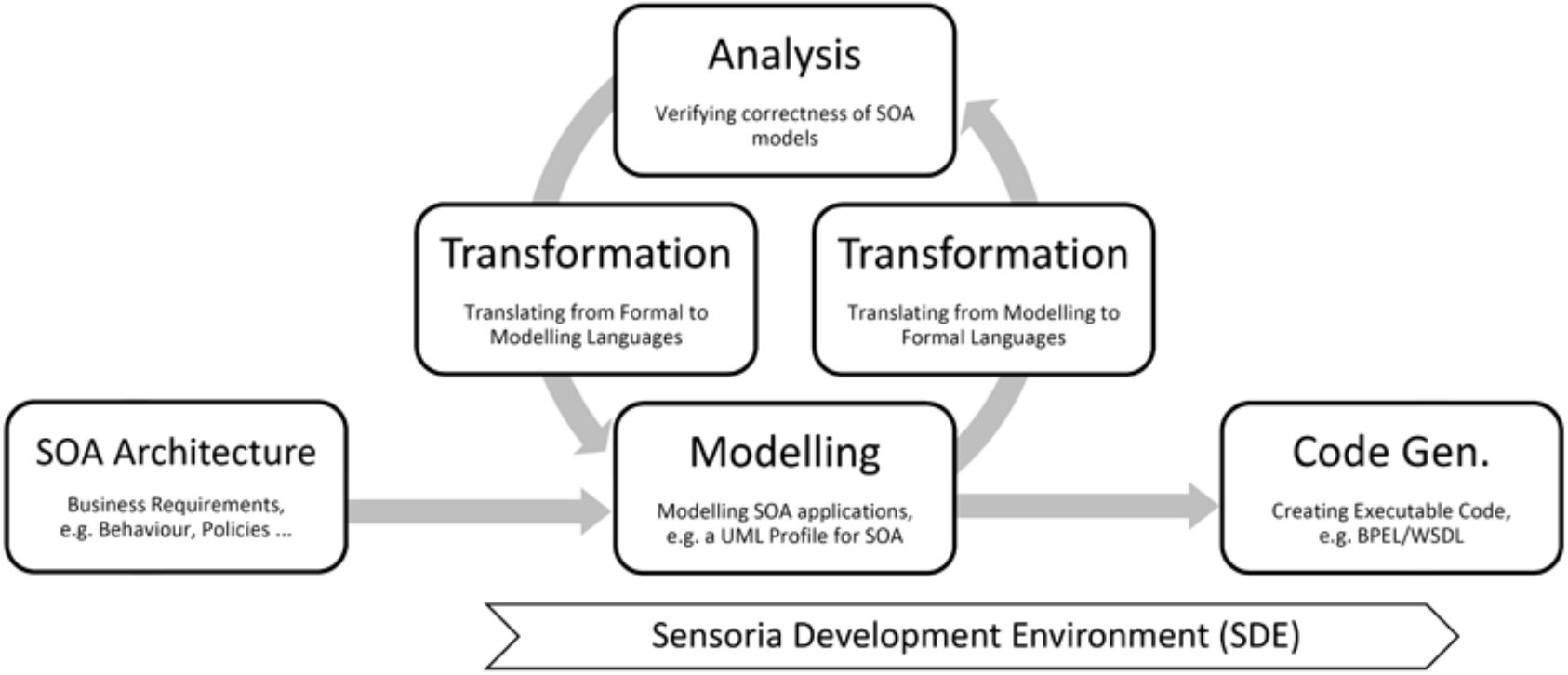
**Blackboard**
Stores data in-between tool function calls

# The SDE and Eclipse

Basic functionality demo

# What can we do with the SDE?

- Back to this diagram:




- The SDE contains tools from several parts of the SOA development process. Let's have a look at what's integrated:

## UML4SOA Profile

(Modelling)

- Aim is simple and easy specification of SOA artefacts in UML
- Focuses on behavioural aspects
- Statical aspects are re-used from soaML.

## SRML

(Modelling)

- The Sensoria Reference Modelling Language is a high-level language for modelling service-oriented systems, that includes a syntax and a mathematical semantics
- SCA-like structure (components, external services, wires)
- declarative description of the business logic based on interactions

## VIATRA2 + UML2WSDL

(Transformation)

- VIATRA2 is a transformation language and framework
- Aim is simple and easy specification of model transformations

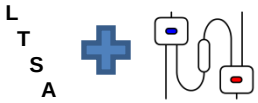
## MDD4SOA Transformers

(Transformation)

- MDD4SOA includes a set of transformers for converting UML4SOA models
- Target languages are BPEL/WSDL, Java, and Jolie

## LTSA/WS-Engineer

(Qualitative Analysis)




- perform model checking on UML diagrams or BPEL code
- Aim is detection of deadlocks or other qualitative properties

## PEPA

(Quantitative Analysis)


- Perform runtime analysis on UML diagrams
- Aim is understanding of the distribution of time spent in the various parts of the program

## LySaTool


(Analysis)

- Tool for verifying security properties of protocols that use cryptography
- Based on protocols modelled in LySa (a process calculus)
- Through program analysis, the LySaTool can guarantee confidentiality and authentication properties.

## CMC - UMC UCTL/Socl model checkers


ISTITUTO DI SCIENZA E TECNOLOGIE DELL'INFORMAZIONE "A. FAEDO"


(Analysis)

- Model checker for systems defined by interacting UML statecharts.
- Allows to model-check on the fly abstract behavioral properties in the Socl braching-time state-action based, parametric temporal logic

## MDD4SOA Analyser

mdd4SOA

(Analysis)

- verifies that an orchestration follows the defined protocol
- Aim is spotting protocol errors before deployment

## DINO

DINO

(Runtime)

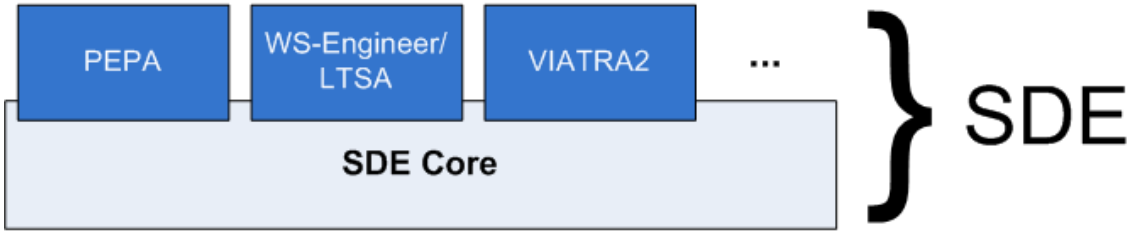
- Dino is a runtime discovery and binding service
- Aim is dynamic selection of services according to properties at runtime

# Using the SDE on a concrete case study

Performing checks in an eUniversity SOA

# How does the SDE work internally?

- Back to the initial ideas about the SDE:




- Questions we have not answered yet:
  - How does the SDE core look like, exactly?
  - How are tools integrated into the SDE?
- Read on...

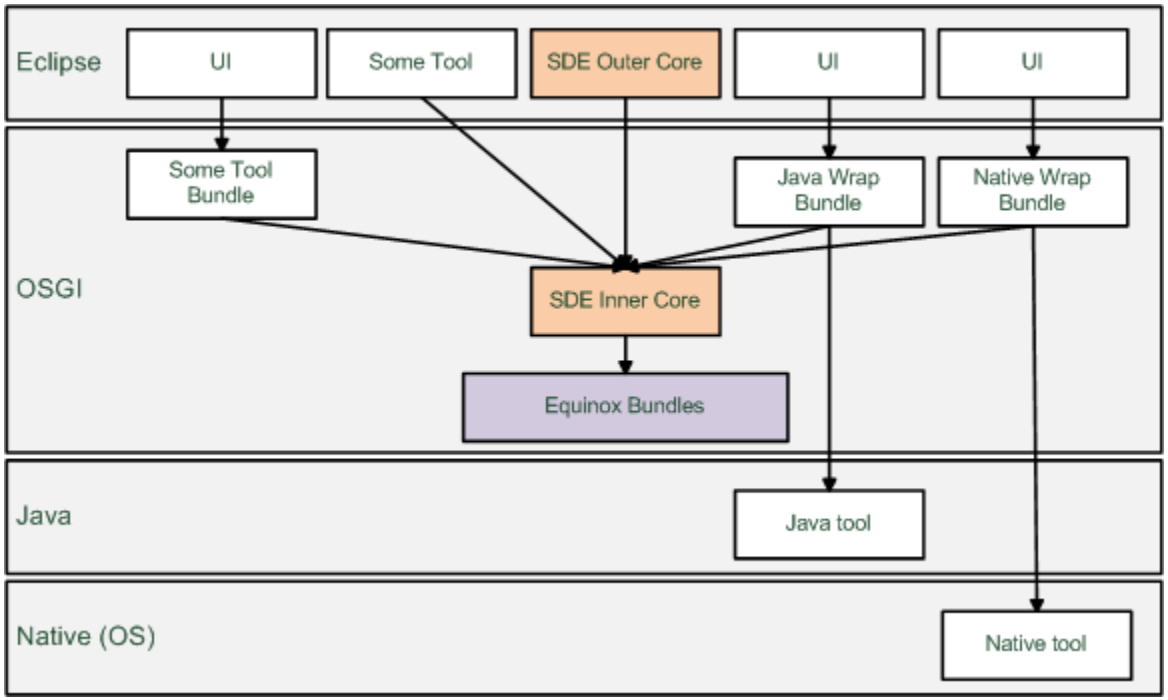
# SDE Architecture

www.sensoria-ist.eu

- The SDE Core is based on Java, OSGi, and Eclipse
  - The **inner core** is based on OSGi only, enabling headless (server-side) usage and orchestrations
  - The **outer core** contains the SDE Core UI, and is based on Eclipse

- OSGI bundles may wrap existing software written in Java or native languages

- UI is usually based on Eclipse (SWT), but may also be written in other GUI frameworks




Information Society Technologies

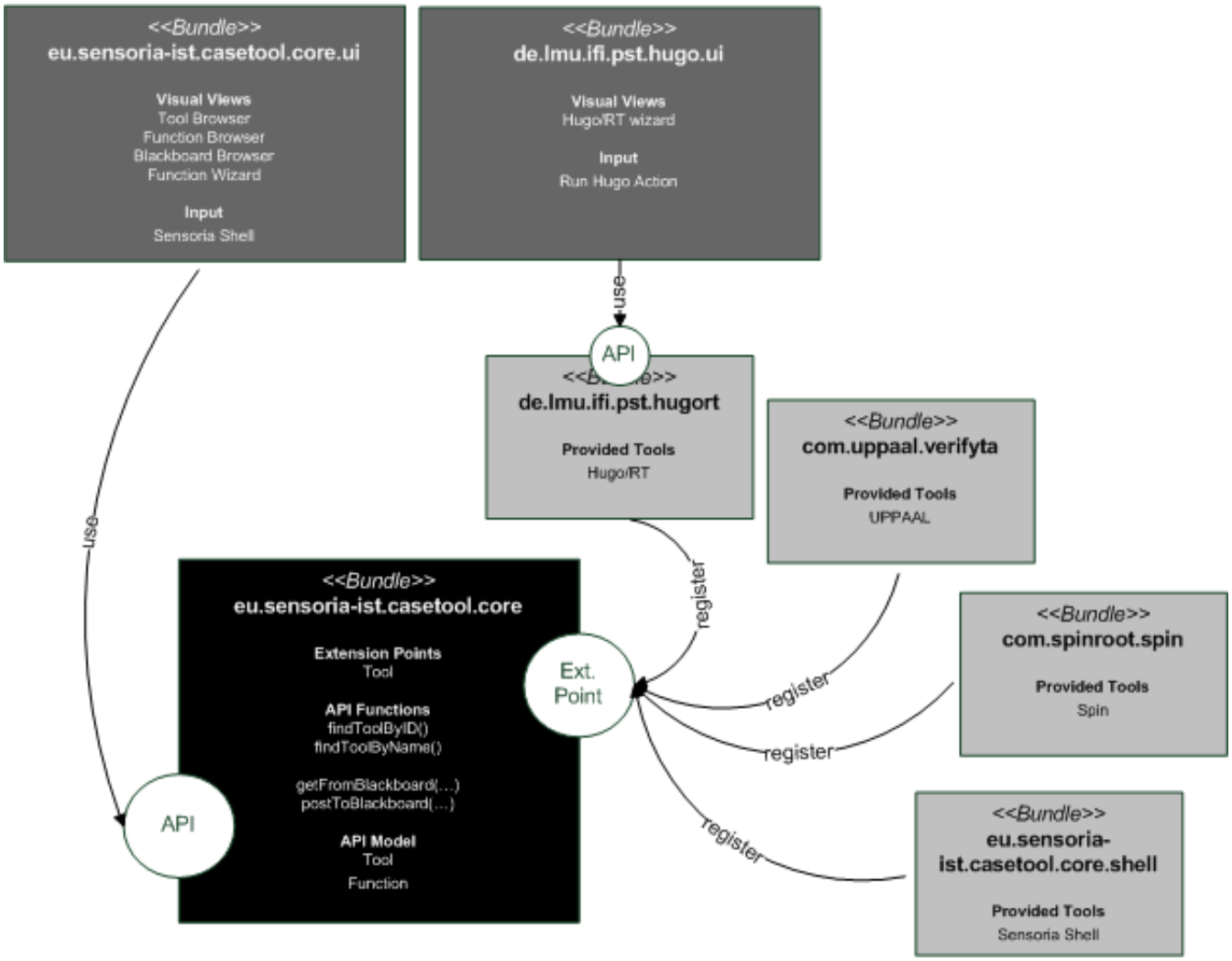
# OSGi + Eclipse



- As Eclipse is OSGi-based, each Plug-In is also an OSGi-Bundle. The difference relates to which dependencies are declared
  - Only OSGi-Dependencies: OSGi Layer
  - Additionally, Eclipse dependencies: Eclipse Layer
- The SDE core consists of 10 OSGi bundles/Eclipse plugins
  - 4 for the core (OSGi-based)
  - 5 for the UI (based on Eclipse)
  - 1 with development helpers (based on Eclipse)
- A bundle may provide
  - **API** (as usual in Java)
  - **Extension points** (new feature of Equinox (OSGi) – defines, in XML, points where other bundles can contribute code).
- We use both mechanisms in the SDE itself

# Core Implementation

www.sensoria-ist.eu



- Interesting Core API functions (Java Interfaces):

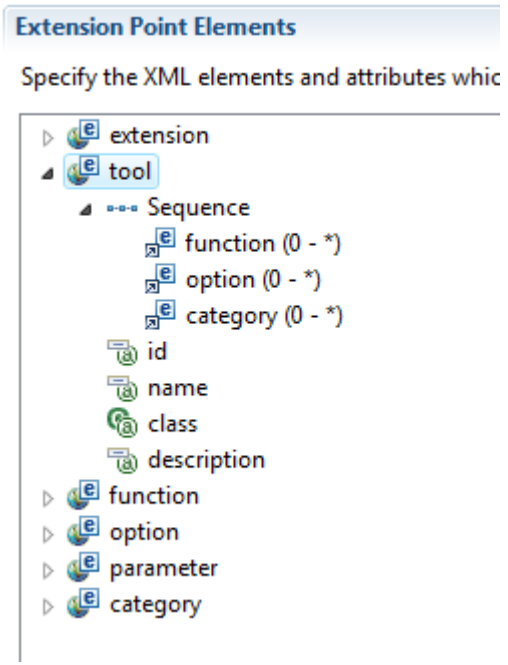
```
public Set<ITool> getTools();
public ITool findToolById(String id);
public void addTool(ITool tool);
...
public void postToBoard(Object object);
public Set<Object> retrieveFromBoardByType(Class<?> clazz);
...
public void setToolOption(String toolId, String option,
                          String value);
...
public void addRemoteCore(String locationURI);
...
```

- These functions can be called from other plug-ins, or through the SDE UI itself.

# Core Extension Points

- The most interesting extension point is for registering **tools**

- On the right, the data structure is shown. A tool needs to be added by specifying
  - a unique ID,
  - a human-readable name,
  - the implementing class,
  - a human-readable description,
  - available functions with parameters,
  - available options,
  - its categories.




- The SDE core parses the extensions on startup and adds all registered tools to the discovery service.

# Annotations

- Instead of writing an extension for the SDE extension point by hand, the metadata can be added to the tool class or interface with annotations.
- Example:

```
@SensoriaTool(name= "UML Problem Analysis Service",
   categories= "Analysis", description= "...")
public class UMLProblemAnalyer {

@SensoriaToolFunction(description= "Function description")
@SensoriaToolFunctionReturns(description= "A string")
public String performSomeAnalysis(
     @SensoriaToolFunctionParameter(description= "...")
          UMLActivity activity);
...
}
```

- The SDE Development Tool can then be used to generate the extension.

# Conclusion

# Conclusion

www.sensoria-ist.eu

- The Sensoria project is working on methods and tools to support developers in creating better service-oriented software
- Therefore, several tools (with formal background) have been developed which users should be able to use in combination.

- The **Sensoria Development (SDE)** was created as a lightweight integration platform into which all these tools can integrate
- It is based on a SOA principle itself and thus views tools as services, which can be discovered, invoked, and composed.
- A graphical composition service is included for easy orchestration.

- Outlook
  - In the lab session, we'll create two SDE tools (and Eclipse plug-ins) from scratch, and integrate them with other tools in the workflow.

Information Society Technologies

# Thank You!