Sensoria

016004

Software Engineering for Service-Oriented Overlay Computers

www.sensoria-ist.eu

D7.4.d

Report on the Sensoria Development Environment (third version)

Lead contractor for deliverable: LMU
Author(s): Philip Mayer (LMU) and István Ráth (BUTE)

Due date of deliverable: February 28, 2010
Actual submission date: January 13, 2010
Revision: Draft
Dissemination level: PU

Contract start date: September 1, 2005 Duration: 48 months
Project coordinator: LMU
Partners: LMU, UNITN, ULEICES, UWARSAW, DTU, PISA, DSIUF,
UNIBO, ISTI, FFCUL, UEDIN, ATX, TILab, FAST, BUTE,
S&N, LSS-Imperial, LSS-UCL, MIP, ATXT, CIR

Integrated Project funded by the
European Community under the
“Information Society Technologies”
Programme (2002—2006)

Executive Summary

Developing service-oriented software involves dealing with multiple languages, platforms, artefacts, and tools. The tasks carried out during development are varied as well; ranging from modelling to implementation, from analysis to testing. For many of these tasks, the SENSORIA project has provided tools aiding developers in their work – with a specific focus on tools based on rigourous formal verification methods, but also including modeling, transformation, and runtime tools. To enable developers to find, use, and combine these tools, we have created a tool integration platform, the SENSORIA Development Environment (SDE), which a) gives an overview of available tools and their area of application, b) allows developers to use tools in a homogeneous way, re-arranging tool functionality as required, and c) enables users to stay on a chosen level of abstraction, hiding formal details as much as possible. In this chapter, we give an in-depth review of the SDE, integrated tools, and ways of using tools in combination for developing and verifying service-oriented software systems.

Contents

1 Introduction

The success of the Service-Oriented Architecture (SOA) [Erl05] in both industry and research has resulted in a growing need for tool support for developers of services and service-based systems. Specific support for developing SOA systems is beneficial in all phases of the development process, ranging from modelling to runtime, from analysis to implementation.

The SENSORIA project [WBC+09] has provided tools and techniques for many of the tasks developers are faced with during the development of SOA systems. A key first result of SENSORIA in this context is a set of languages for describing SOA systems, like, for example, the UML profile for services (UML4SOA) [MSK08] and accompanying tool support. However, the main consideration in SENSORIA was rigourous engineering of service-oriented systems with a specific focus on formal verification. As our verification and validation methods are often directly based on a formal model, tool support had to be created for allowing developers to use these methods while staying on their chosen level of abstraction – for example, UML. To deal with this issue, SENSORIA has investigated model transformations, a concept taken from the model-driven architecture (MDA) community, to ease the transition between developer-level models of a SOA system and the formal languages required for verification, with the additional benefit of being able to generate executable code as well. Finally, runtime support for services in the form of dynamic discovery mechanisms requires a broker infrastructure and testing tools which should be accessible during development as well.

Altogether, these considerations have led us to develop a tooling platform, the SENSORIA Development Environment (SDE) [MRH08], which integrates the various tools required in the service development process, including modelling, analysis, code generation, and runtime functionality. The SDE:

a) gives an overview of available tools and their area of application,

b) allows developers to use tools in a homogeneous way, re-arranging tool functionality as required, and

c) enables users to stay on a chosen level of abstraction, hiding formal details as much as possible.

In this chapter, we give an in-depth review of the SDE, integrated tools, and ways of using tools in combination for developing and verifying service-oriented software systems. In section 2, we give a high-level overview of the SDE. Section 3 further details design and implementation of our integration platform. In section 4, we give an overview of tools integrated into the SDE. Section 5 shows examples of how tools can be orchestrated to perform in collaboration. Finally, section 6 concludes the chapter.

2 High-Level Overview

The SENSORIA project aims to support developers of service-oriented software systems at various points in the development process. Specific focus is placed on (formal) verification of service artefacts, which includes appropriate modelling support for developers as well as code generation and runtime support. Through various tools, we are thus able to offer functionality which covers the complete model-driven process of service engineering, which is shown in Fig. 1.

After starting with requirements for a SOA-based system, developers advance to the modelling phase. From this phase, various analyses of the models may be performed, many of them carried out with the help of automated model transformations. Finally, code is generated from the improved models; runtime support is available for executing this code on various platforms. The figure shows the phases which are covered by tools integrated into the SDE – Modelling, Transformation, Analysis, Code Generation, and Runtime. The following functionality is available in each of these phases:

- **Modelling.** Graphical editors for familiar modelling languages such as UML, which allow intuitive modelling at a high abstraction level, and also text- and tree-based editors for formal languages like process calculi.

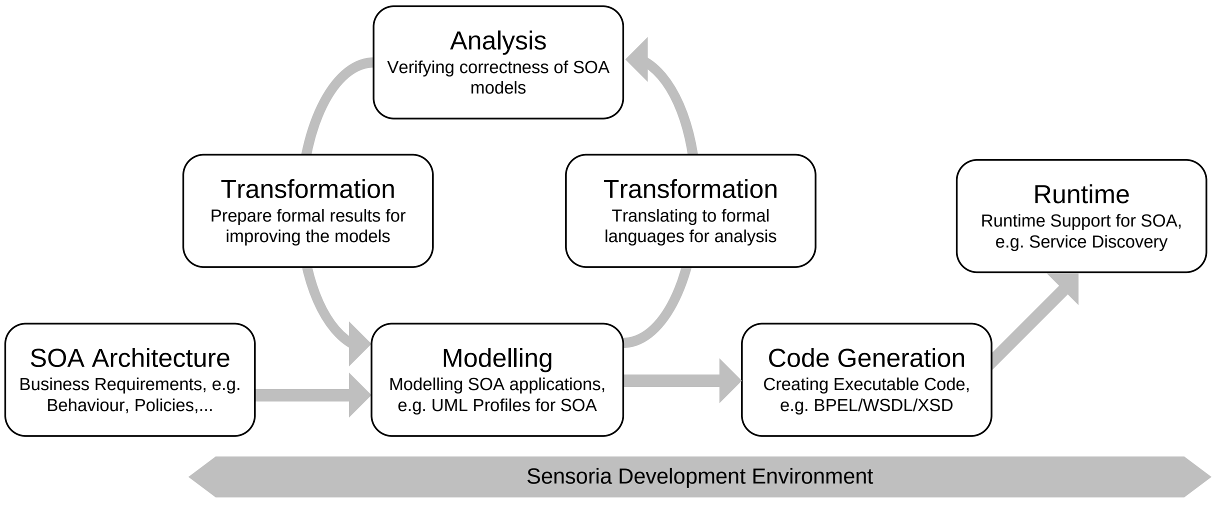

Analysis
Verifying correctness of SOA models
Transformation
Prepare formal results for improving the models
Transformation
Translating to formal languages for analysis
Runtime
Runtime Support for SOA, e.g. Service Discovery
SOA Architecture
Business Requirements, e.g. Behaviour, Policies,...
Modelling
Modelling SOA applications, e.g. UML Profiles for SOA
Code Generation
Creating Executable Code, e.g. BPEL/WSDL/XSD
Sensoria Development Environment

Figure 1: Development Approach

- **Model Transformation Functionality, including Code Generation.** Automated model transformations from UML to process calculi and back to bridge the gap between these worlds; also, generation of executable code (for example, Web Service standards like BPEL).
- **Formal Analysis Functionality.** Model checking and numerical solvers for stochastic methods based on process calculi code defined by the user or generated by model transformation.
- **Runtime Functionality.** Integration of runtime platforms, for example BPEL process engines or the Java runtime as well as runtime support for services, for example dynamic service brokering.

The functionality indicated in the previous list is implemented in various tools, some of which have been developed within SENSORIA, some developed outside of the project (for a full list of SENSORIA tools, see section 4). The tools are not only developed at different sites, but are also vastly different with regard to user interface, functionality, required computing power, execution platform and programming language. However, all of the tools contribute to the development process and in many cases deliver artefacts which may serve as input to other tools.

The SENSORIA Development Environment (SDE) provides this functionality through a carefully designed, lightweight integration architecture. This is achieved through the following core features:

- **A SOA-based platform.** The SDE itself is based on a Service-Oriented Architecture, allowing easy integration of tools and querying the platform for available functionality. The tools hosted in the SDE are installed and handled as services.
- **A Composition Infrastructure.** As development of services is a highly individual process and may require several steps and iterations, the SDE offers a composition infrastructure which allows developers to automate commonly used workflows as an orchestration of integrated tools.
- **Hidden Formal Methods.** To allow developers to use formal tools without requiring them to understand the underlying formal semantics, the SDE encourages the use of automated model transformations which translate between high-level models and formal specifications.

As with services in a SOA, tool composition in our integration tool is a lightweight one, i.e., the connection between tools is not a priori fixed and adding additional tools requires only minimal change to the integrated tools. Using the tool-as-a-service metaphor, tools are services, each consisting of functions which can be invoked by the user or other services. Contrary to Web services [WCL$^+$05], user interaction is very important for some software development tools. For example, a modelling tool requires a lot of user interaction – ideally, the modelling tool runs on the computer of the user. A model checker, on the other hand, requires a lot of computing power and thus will most likely run on a dedicated server to be accessed remotely with none or only a minimal, generated UI available. Both use cases are supported in the SDE.

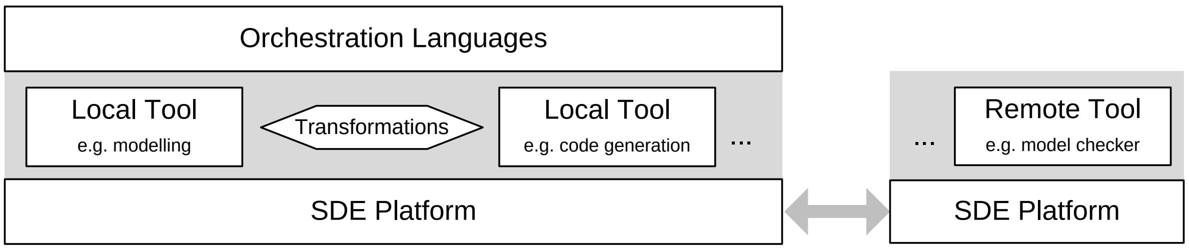


Figure 2: SDE Architecture

By using a SOA-based infrastructure, combining tools into more complex tool chains is straightforward, i.e. possible via dedicated orchestration languages. A typical scenario for tool composition can be found in the analysis and verification of software; for example, model checkers require a certain input format into which most source models first need to be transformed; the same applies to the output. The SDE contains both a textual (JavaScript) and a graphical (UML-based) orchestration language, allowing users to integrate various tools, thereby handling the data flow between these tools. Having encapsulated the integrating steps, they can be run over and over again for performing the same steps with different input and output data.

Finally, the SDE aims at providing formal verification tools to pragmatic developers. This requires, as indicated above, the use of model transformations to allow developers to stay on their chosen level of abstraction while still enjoying the results available through rigourous verification methods. Through tool chaining and the ability to install verification tools remotely, the SDE enables an MDA-like approach to the analysis of service artefacts.

Fig. 2 shows the architecture of the SDE. As discussed previously, the integration platform hosts a number of tools as services. Through its dedicated orchestration infrastructure, the SDE allows developers to orchestrate tools to be used in combination, which includes using model transformations and a remote invocation functionality for invoking tools hosted on different machines.

The next section will introduce the technical details of the SDE implementation.

3 Design and Implementation

The aim of SENSORIA is to support the creation of service-oriented software by augmenting existing development processes and tools. A requirement for the SDE was therefore to integrate with existing tools and platforms for the development of SOA systems. For this reason, the SDE is based on the well-known Eclipse platform [Ecl09b] and its underlying, service-oriented OSGi [OSG08] framework. OSGI is based on so-called bundles, which are components grouping a set of Java classes and meta-data providing among other things name, description, version, exported and imported packages of the bundle. A bundle may provide arbitrary services to the platform.

3.1 SDE Core and UI

The technical architecture of the SDE is depicted in Fig. 3, which shows the SDE Platform as an OSGi bundle, its dependencies and dependent bundles.

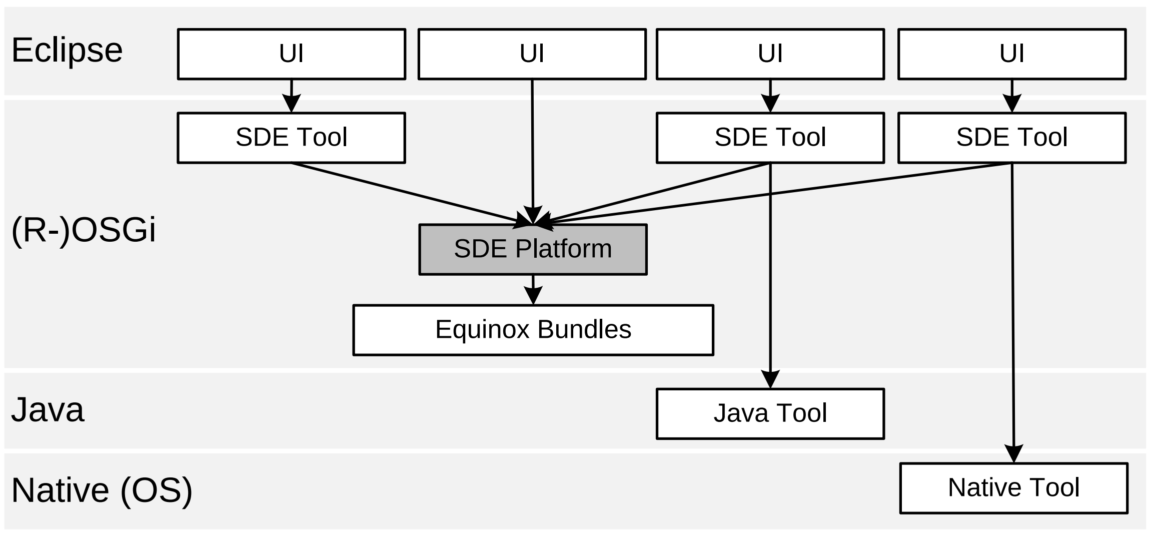


Figure 3: SDE Technical Architecture

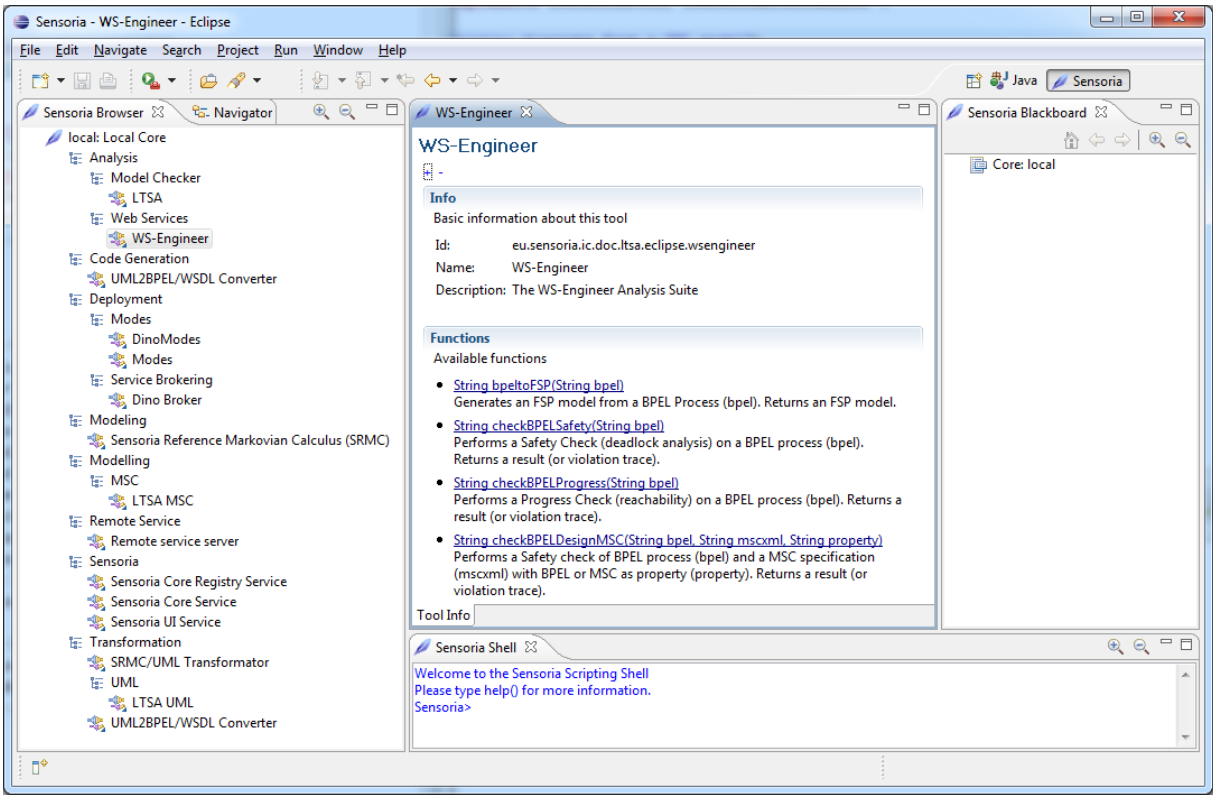
Fundamentally, all tools are integrated as OSGi bundles which offer certain *functions* for invocation by the platform. As indicated above, the tools integrated into the SDE are vastly different, ranging from user-driven graphical modelling tools to computationally intensive analysis tools with very basic interaction mechanisms. Thus, it is not possible to define a common API for all tools. In the SDE, this problem is solved by using (declarative) OSGi services for each tool. Furthermore, the SDE allows tools to provide their own UI, but also provides a generic invocation mechanism which enables users to invoke arbitrary functions, either directly or through an orchestration. Finally, tool integration requirements should be kept low to ensure integration of as many tools as possible. The SDE re-uses OSGi and Eclipse technology and declarative service descriptions which are generated from Java annotations for a fast and straightforward integration process.

As can be seen in Fig. 3, the SDE platform and the integrated tools are based on (R-)OSGi only (or, more specifically, the Equinox implementation of OSGi [Ecl09a]). This means that fundamentally, tools must be implemented in Java, although they may wrap native code or remote invocations as they wish. Being only based on OSGi, they can be invoked completely independently from Eclipse. If they additionally choose to provide a UI, this UI is integrated into and based on the Eclipse platform, as is the UI for the SDE platform itself.

Fig. 4 shows a screenshot of the SDE UI. On the left hand side, the tool browser shows installed tools available for invocation and automation. Tools are grouped by category, allowing quick access by application area. Double-clicking a tool in the browser yields more information about the tool and its functionality. This information is shown in the view in the middle: As an example, an integrated tool for qualitative analysis (WS-Engineer) is shown in more detail. Each tool function displayed here can be invoked by clicking the link and providing the parameters. Finally, on the right, the SENSORIA Blackboard is shown, which is a storage area where tools may place arbitrary objects for later use. Finally, at the bottom, the SENSORIA Shell is displayed, which is a live JavaScript execution environment (see section 3.2).

As an example for a function invocation, clicking on the `bpelToFSP()` function in the WS-Engineer tool yields the following dialogs, where the data for the single parameter (`bpel`) can be selected from various sources (Fig. 5).

Finally, the SDE core integrates with R-OSGi [RAR07] to provide the ability to host tools for external invocation, and connect to remote SDE cores. The tools in the tool view in Fig. 4 (left), for example, are listed under the local core. Further (remote) cores may be added as required, and their tools are



WS-Engineer

Info
Basic information about this tool

Id: eu.sensoria.ic.doc.ltsa.eclipse.wsengineer
Name: WS-Engineer
Description: The WS-Engineer Analysis Suite

Functions
Available functions

- String bpeltoFSP(String bpel)
  Generates an FSP model from a BPEL Process (bpel). Returns an FSP model.
- String checkBPELSafety(String bpel)
  Performs a Safety Check (deadlock analysis) on a BPEL process (bpel). Returns a result (or violation trace).
- String checkBPELProgress(String bpel)
  Performs a Progress Check (reachability) on a BPEL process (bpel). Returns a result (or violation trace).
- String checkBPELDesignMSC(String bpel, String mscxml, String property)
  Performs a Safety check of BPEL process (bpel) and a MSC specification (mscxml) with BPEL or MSC as property (property). Returns a result (or violation trace).

```
Welcome to the Sensoria Scripting Shell
Please type help() for more information.
Sensoria>
```

Figure 4: SDE Screenshot

then listed and used in the same way as described above. Furthermore, the blackboard (right) also distinguishes between the various cores.

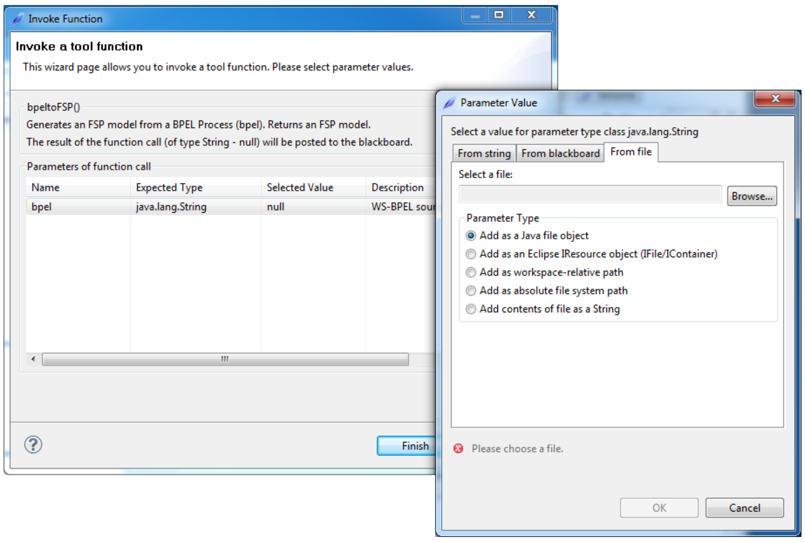
3.2 Composing Tools

The SDE provides the ability to compose new tools out of existing ones, a process known as orchestration in the SOA world. Creating orchestrations is possible using two mechanisms: A textual, JavaScript-based approach, and a graphical, UML-activity-diagram-like workflow approach.

3.2.1 Orchestrating with JavaScript

The ability to use tool APIs directly within JavaScript enables developers to create a workflow by simply invoking tool functions and passing data in-between those functions. To enable the newly created workflow to be usable as a tool in its own right, two things are required: Instead of simply creating a workflow, a JavaScript function definition is required which states a function name and parameters. As each tool, function, parameters, and return types may have descriptions and additional metadata attached, this metadata must be specified in some way in the JavaScript source files. Both points have been addressed in the SDE. The first is simple; function definitions are already part of the JavaScript specification. The second was solved by employing a JavaDoc-comment-style approach to metadata specification. Tags like `@description` are used to convey metadata information.

As an example, Fig. 6 (left) shows a script for converting UML2 activity diagrams to BPEL, then analysing them using the WS-Engineer tool, and finally converting the result back to UML2 sequence diagrams showing the error trace. Fig. 6 (right) shows the converted tool inside the SDE tool browser. Scripts created like this can be used on any SDE installation which has the required tools installed. No particular deployment is necessary save copying the script and registering it with the core.


Invoke Function

Invoke a tool function

This wizard page allows you to invoke a tool function. Please select parameter values.

bpeltoFSP()

Generates an FSP model from a BPEL Process (bpel). Returns an FSP model.

The result of the function call (of type String - null) will be posted to the blackboard.

Parameters of function call

Name	Expected Type	Selected Value	Description
bpel	java.lang.String	null	WS-BPEL sour

Parameter Value

Select a value for parameter type class java.lang.String

From string | From blackboard | From file

Select a file:

Parameter Type

- [✓ Add as a Java file object]
- Add as an Eclipse IResource object (IFile/IContainer)
- Add as workspace-relative path
- Add as absolute file system path
- Add contents of file as a String

Please choose a file.

Figure 5: SDE Wizard

For testing purposes, the SDE also contains a JavaScript live execution environment, the SDE Shell (Fig. 4), where JavaScript commands can be executed without compiling a complete script.

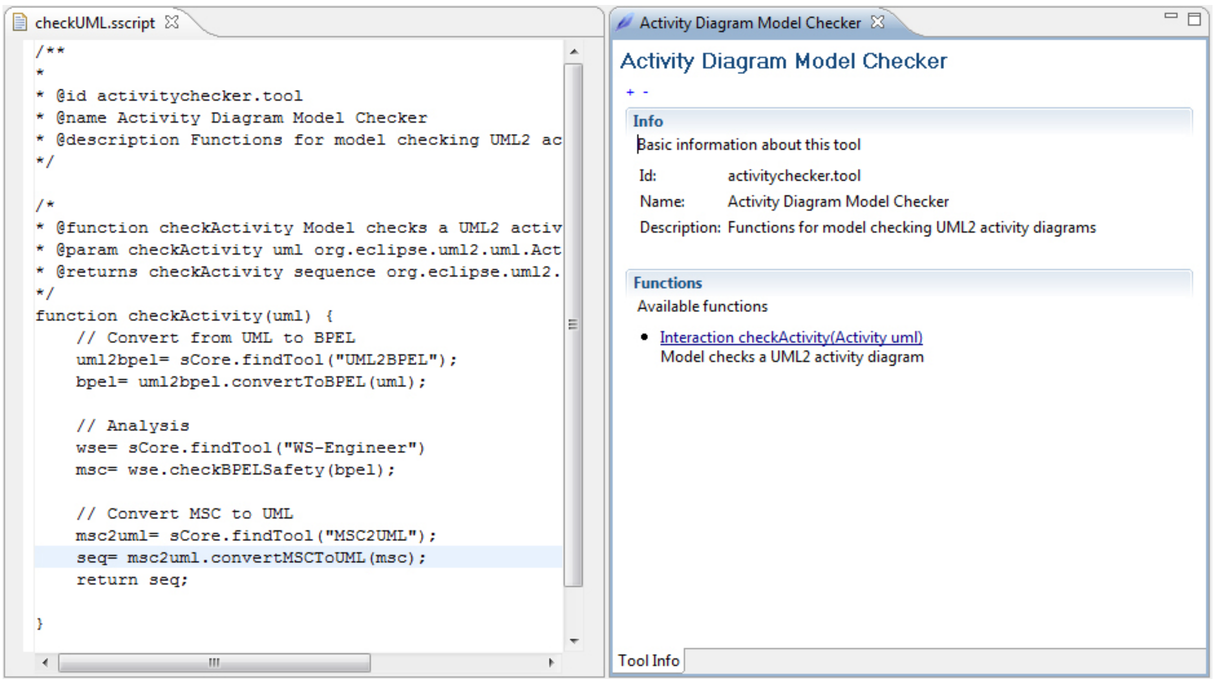
3.2.2 Graphical Orchestration

Besides the ability to use JavaScript for orchestration as indicated above, the SDE also contains the ability to orchestrate tools graphically. The syntax used is that of UML2 activity diagrams, where the main focus is on data flow, i.e. the flow of information from pin to pin. An activity in the diagram represents one function in the tool to be generated which has input pins (parameters) and one output pin (return type). Inside the activity, actions represent function calls to arbitrary (installed) tools. These actions have pins themselves; data flow edges model the data transfer.

As an example, consider the screenshot in Fig. 7, which shows the orchestration introduced in the previous paragraph as a graphical workflow, including the editor which supports it. The function `checkActivity(uml)` is modelled as an UML2 activity, and each call to a particular function of an installed tool is modelled as an action. On the right-hand side, the palette shows all available tools and the functions they provide. Once modelled, an orchestration such as the one above is converted to a Java class, compiled in-memory and installed as a tool in the SDE.

3.3 Extending the Platform

The SOA-based architecture of the SDE makes it easy to add new tools – the SDE publishes a core API and an extension point for registering tools. Basically, each tool is an OSGi bundle with some published API and metadata XML to register the tool with the SDE core. Thus, creating a facade class and registering the class with the SDE extension point enables tool functionality to be immediately available within the SDE, both for manual invocation and automation. Tools within the SDE are loosely coupled, as they are fundamentally independent from each other and interact through their published service interfaces only. They may, of course, require other tools to be installed for them to work. This is defined in a declarative way through the Equinox extension mechanism and checked by the platform prior to tool installation. The SDE core also contains a set of Java 5 annotations, which enable tool developers to define their tools and functions without writing any XML. As an example, consider Fig. 8: On the left-hand side, a tool interface with SDE annotations is shown; on the right-hand side, the corresponding tool view in the SDE.



```
/**
*
* @id activitychecker.tool
* @name Activity Diagram Model Checker
* @description Functions for model checking UML2 ac
*/

/*
* @function checkActivity Model checks a UML2 activ
* @param checkActivity uml org.eclipse.uml2.uml.Act
* @returns checkActivity sequence org.eclipse.uml2.
*/
function checkActivity(uml) {
    // Convert from UML to BPEL
    uml2bpel= sCore.findTool("UML2BPEL");
    bpel= uml2bpel.convertToBPEL(uml);

    // Analysis
    wse= sCore.findTool("WS-Engineer")
    msc= wse.checkBPELSafety(bpel);

    // Convert MSC to UML
    msc2uml= sCore.findTool("MSC2UML");
    seq= msc2uml.convertMSCToUML(msc);
    return seq;

}
```

Activity Diagram Model Checker

Info
Basic information about this tool

Id:	activitychecker.tool
Name:	Activity Diagram Model Checker
Description:	Functions for model checking UML2 activity diagrams

Functions
Available functions

- Interaction checkActivity(Activity uml)
  Model checks a UML2 activity diagram

Figure 6: Orchestration with JavaScript

The API defined within the integration tool service bundle provides access to all installed tools. A tool may use this API to verify installation of required tools; search for tools based on meta-data, and invoke functionality as needed. Therefore, it serves as a discovery service which moderates between the tools. Once the connection has been made, communication between tools is done directly.

4 Integrated Tools

This chapter lists all tools which have been integrated into the SDE platform, sorted by integrated category.

4.1 Modelling

ArgoUML ArgoUML is an open source UML modelling tool which includes support for all standard UML 1.4 diagrams.

`http://argouml.tigris.org/`

Rational Software Architect Rational Software Architect is a UML modelling tool which supports UML2.0 profiles and is built on the Eclipse platform.

`http://www.ibm.com/software/awdtools/architect/swarchitect/`

MagicDraw MagicDraw is a platform independent UML modeller with profile support for UML2.

`http://www.magicdraw.com/`

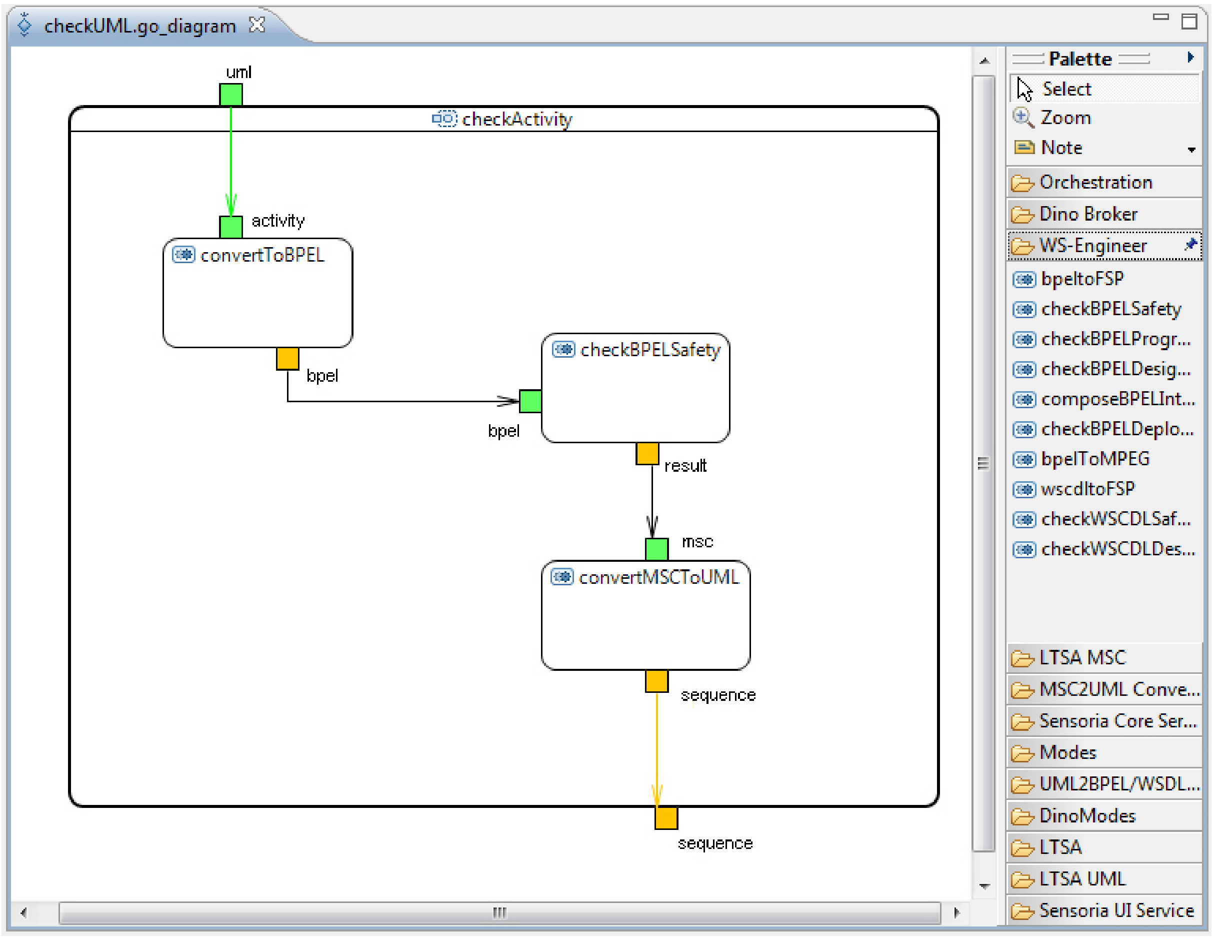


Figure 7: Graphical Orchestration

4.2 Transformation and Deployment

Hugo/RT Hugo/RT is a UML model translator for model checking, theorem proving, and code generation: A UML model containing active classes with state machines, collaborations, interactions, and OCL constraints can be translated into the system languages of the real-time model checker UPPAAL, the on-the-fly model checker SPIN, the system language of the theorem prover KIV, and into Java and SystemC code.

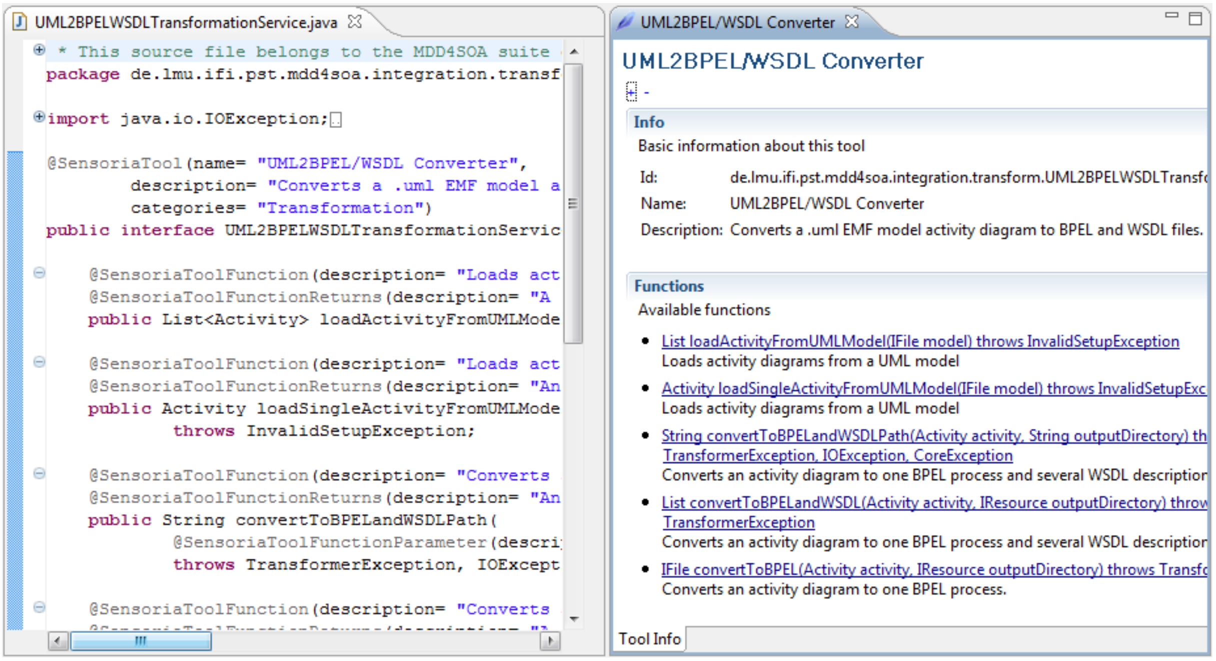
`http://www.pst.informatik.uni-muenchen.de/projekte/hugo/`

VIATRA2 The main objective of the VIATRA2 (VIsual Automated model TRAnsformations) framework is to provide a general-purpose support for the entire life-cycle of engineering model transformations including the specification, design, execution, validation and maintenance of transformations within and between various modelling languages and domains.

`http://wiki.eclipse.org/VIATRA2`

SOA2WSDL-Transformation The SOA2WSDL transformation, written in VIATRA2, takes high level UML models and produces WSDL (Web Services Description language) output.

`http://viatra.inf.mit.bme.hu/`



```
 * This source file belongs to the MDD4SOA suite
package de.lmu.ifi.pst.mdd4soa.integration.transf

import java.io.IOException;

@SensoriaTool(name= "UML2BPEL/WSDL Converter",
        description= "Converts a .uml EMF model a
        categories= "Transformation")
public interface UML2BPELWSDLTransformationServic

    @SensoriaToolFunction(description= "Loads act
    @SensoriaToolFunctionReturns(description= "A
    public List<Activity> loadActivityFromUMLMode

    @SensoriaToolFunction(description= "Loads act
    @SensoriaToolFunctionReturns(description= "An
    public Activity loadSingleActivityFromUMLMode
            throws InvalidSetupException;

    @SensoriaToolFunction(description= "Converts
    @SensoriaToolFunctionReturns(description= "An
    public String convertToBPELandWSDLPath(
            @SensoriaToolFunctionParameter(descri
            throws TransformerException, IOExcept

    @SensoriaToolFunction(description= "Converts
```

UML2BPEL/WSDL Converter

Info

Basic information about this tool

Id:	de.lmu.ifi.pst.mdd4soa.integration.transform.UML2BPELWSDLTransf
Name:	UML2BPEL/WSDL Converter
Description:	Converts a .uml EMF model activity diagram to BPEL and WSDL files.

Functions

Available functions

- List loadActivityFromUMLModel(IFile model) throws InvalidSetupException
  Loads activity diagrams from a UML model
- Activity loadSingleActivityFromUMLModel(IFile model) throws InvalidSetupExc
  Loads activity diagrams from a UML model
- String convertToBPELandWSDLPath(Activity activity, String outputDirectory) th
  TransformerException, IOException, CoreException
  Converts an activity diagram to one BPEL process and several WSDL descriptior
- List convertToBPELandWSDL(Activity activity, IResource outputDirectory) throw
  TransformerException
  Converts an activity diagram to one BPEL process and several WSDL descriptior
- IFile convertToBPEL(Activity activity, IResource outputDirectory) throws Transfo
  Converts an activity diagram to one BPEL process.

Figure 8: SDE Tool Registration

SRMC/UML Bridge The SRMC/UML bridge offers facilities for meta-model transformation. It translates a subset of UML2 models (Interactions and State Machines) into an SRMC description for performance evaluation. Results are reflected back into the UML model.

`http://groups.inf.ed.ac.uk/srmc/`

UML2PEPA Transformation The UML2PEPA transformation, written in VIATRA2, takes high level UML models and produces PEPA models used for analysis in the PEPA/SRMC tool.

`http://viatra.inf.mit.bme.hu/`

Modes Parser and Browser The Modes Parser and Browser is a WS-Engineer plug-in to parse and extract broker requirements from UML2 Modes Models.

`http://www.doc.ic.ac.uk/ltsa/eclipse/wsengineer`

4.3 Analysis

LTSA LTSA is a verification tool for concurrent systems. It checks that the specification of a concurrent system satisfies the properties required of its behaviour. In addition, LTSA supports specification animation to facilitate interactive exploration of system behaviour.

`http://www.doc.ic.ac.uk/ltsa/`

WS-Engineer The LTSA WS-Engineer plug-in is an extension to the LTSA Eclipse Plug-in which allows service models to be described by translation of the service process descriptions, and can be used to perform model-based verification of Web service compositions.

`http://www.doc.ic.ac.uk/ltsa/eclipse/wsengineer/`

SRMC Core SRMC (SENSORIA Reference Markovian Calculus) Core provides support for SRMC, an extension to PEPA. It covers steady-state analysis of the underlying Markov chain of SRMC descriptions.

`http://groups.inf.ed.ac.uk/srmc/`

MDD4SOA Protocol Analysis The MDD4SOA Protocol Analysis Tool verifies protocol compliance of a service orchestration denoted in UML4SOA given a protocol state machine. The verification yields a violation graph in case of an error.

`http://www.mdd4soa.eu/`

SPIN Spin is a popular open-source software tool, used by thousands of people worldwide that can be used for the formal verification of distributed software systems.

`http://spinroot.com/`

UPPAAL Uppaal is an integrated tool environment for modelling, validation and verification of real-time systems modelled as networks of timed automata, extended with data types (bounded integers, arrays, etc.).

`http://www.uppaal.com/`

CMC / UMC CMC and UMC are model checkers and analysers for systems defined by interacting UML state charts. Both allow on-the-fly model checking of abstract behavioural properties in the Socl branching-time state-action based, parametric temporal logic.

`http://fmt.isti.cnr.it/cmc/`,`http://fmt.isti.cnr.it/umc/`

LySa tool LySa is a static analyser for security protocols defined in the LYSA process calculus. The tool provides a LYSA editor to assist users in the modelling of protocols. Given a LYSA model the analyser will verify properties related to secrecy and authentication.

`http://www2.imm.dtu.dk/cs_LySa/lysatool/`

4.4 Deployment and Runtime

MDD4SOA Transformers The MDD4SOA transformers are a set of EMF transformers for converting UML4SOA models into target languages. Supported are BPEL/WSDL, Java, and Jolie.

`http://www.mdd4soa.eu/`

UML2AXIS Transformation The UML2AXIS transformation, written in VIATRA2, takes high level UML models and produces Web service code based on the Apache Axis Java library.

`http://viatra.inf.mit.bme.hu/`

Dino Broker The Dino Broker provides dynamic runtime discovery of services which are described in OWL and WSDL documents, thus enabling developers to bind services which correspond to specific criteria.

`http://www.cs.ucl.ac.uk/staff/a.mukhija/dino/`

5 Tool Applications

The tools listed in the previous chapter can be combined in various ways to achieve different transformations and analyses. Fig. 9 lists, non-exhaustively, the links between the tools.

As examples, we provide three scenarios with different tools to give some insights into how tools have been chained together within the SENSORIA project. In the following sections, we use four paragraphs to describe each scenario:

- **Use Case** describes when and why to use a certain tool chain.
- In **Tools Involved**, we list the tools required to perform the functionality of the scenario.
- **Data Flow** shows the individual steps to be executed in the tool chain.
- Finally, **Results** describes the consequences and benefits of using the scenario.

The tool chains may be realised manually, i.e. with the user performing one step after another and storing the intermediate objects on disk or on the blackboard, or automatically by employing the JavaScript orchestrator or the graphical orchestration mechanism.

5.1 Checking and Deploying Service Orchestrations

Use Case Using a model-driven approach for developing software has been advocated for some time. SENSORIA addresses this area with a customised UML2 profile for modelling SOAs, and in particular, service orchestrations. Besides modelling the orchestration implementation itself, a behavioural protocol can help to assess the external behaviour of the orchestration and used to verify the actual implementation. Once a service orchestration has been verified, it needs to be transformed to code in target languages like BPEL or Java to deploy it for execution.

Tools Involved This tool chain includes a UML modeler with profile support, like MagicDraw or Rational Software Architect. A protocol analysis tool (part of MDD4SOA) is used to report on protocol violations. Finally, model transformers (also part of MDD4SOA) are used to transform the UML specifications to code in executable languages (for example, BPEL and WSDL) for deployment.

Data Flow The chain starts with the user who employs a UML modeler to design both the orchestration implementation and the service protocol. The resulting diagrams are saved as documents in the XMI format. These files can then be used by the MDD4SOA Protocol Analyser, which either reports no protocol violations or creates a violation trace. This process is repeated until the process is error-free. Finally, the UML2 models are read by the MDD4SOA Transformers, which generate the appropriate target code, depending on which language has been selected by the user.

Results Chaining tools together in this fashion enables the developer to quickly react to changes in requirements, as the chain can be run automatically whenever a change has occurred, either informing the user of newly introduced problems in the protocol or, if the protocol is valid, with the new implementation in the selected target language.

5.2 Qualitative and Quantitative Analysis

Use Case Service-oriented software systems are commonly distributed – they make use of a network to combine various individual software components to work in coordination to reach a higher-level goal. In general, a SOA system contains many different threads of execution, which run in parallel and interact with one another in nontrivial ways. This poses a difficult problem to software designers, as the

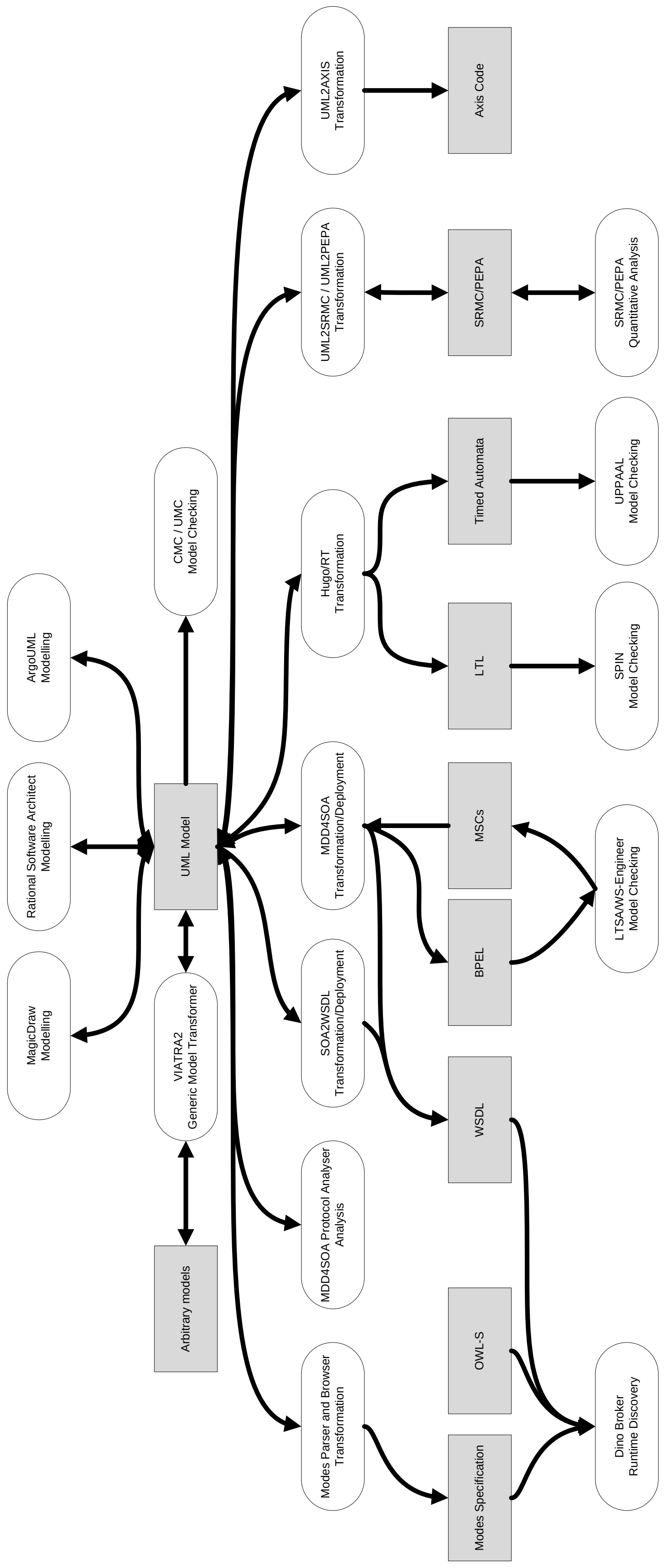

MagicDraw Modelling
Rational Software Architect Modelling
ArgoUML Modelling
VIATRA2 Generic Model Transformer
Arbitrary models
UML Model
CMC / UMC Model Checking
Modes Parser and Browser Transformation
MDD4SOA Protocol Analyser Analysis
SOA2WSDL Transformation/Deployment
MDD4SOA Transformation/Deployment
Hugo/RT Transformation
UML2SRMC / UML2PEPA Transformation
UML2AXIS Transformation
Modes Specification
OWL-S
WSDL
BPEL
MSCs
LTL
Timed Automata
SRMC/PEPA
Axis Code
Dino Broker Runtime Discovery
LTSA/WS-Engineer Model Checking
SPIN Model Checking
UPPAAL Model Checking
SRMC/PEPA Quantitative Analysis

Figure 9: Tool Chaining in the SDE

interaction of such threads needs to be analysed in order to ensure that no undesirable effects (such as deadlocks) occur. Furthermore, it is not always clear how the system time is spent during runtime. Therefore, mechanical checkers are needed to verify whether a certain implementation is free from conditions such as deadlocks, and secondly for assessing the runtime characteristics of the overall system.

Tools Involved Again, we employ UML modelers like Rational Software Architect or MagicDraw for the modelling of a service-oriented system written in UML. Based on these models, quantitative analysis as well as qualitative analysis is then performed by the SRMC tool and the WS-Engineer tool, respectively. While the former is able to deal with UML directly, the latter requires the BPEL format as input, so we bring in another tool (one of the MDD4SOA transformers) for converting between UML and BPEL.

Data Flow The chain starts with the user who employs a UML modeler to design a model of communicating systems in UML2. The resulting model, in the format of an UML2 XMI file, can be read directly by the SRMC tool to report on the distribution of time spent in the various states of the process. Using the MDD4SOA transformers, the UML2 model is converted to BPEL to serve as input for WS-Engineer, which is used to verify the required properties (for example, freeness from dead-locks). Finally, the result of the analysis is shown to the user: The quantitative analysis can be directly annotated to the original UML model (or output as graphs), the qualitative analysis – if resulting in an error trace – is shown as Message Sequence Charts (MSCs) or UML2 sequence diagrams.

Results This tool chain provides the user with a "one-click" verification of the model – instead of requiring the user, as is common in many verification tools, to activate a translation of service implementations, feed the translation through a model parser, compile the model, and invoke a verify option on the model checker. All these single steps are handled by the tool chain and the script used to combine the two different analysers. Thus, checking becomes less of a hassle and will be executed more often, resulting in higher-quality systems.

5.3 Modes-Based Dynamic Runtime Discovery

Use Case One of the promises of the Service-Oriented Architecture is the ability to quickly react to changes, for example – on the business level – a change of a business partner, or – on a technical level – network connection problems or server overload. To deal with these problems, the concept of dynamic service discovery and binding has been introduced, which enables developers to specify, on an abstract level, the properties and constraints required of certain services needed by an orchestration. Specification of such properties, the criteria of when to change the service to be used (specified by "modes"), and testing of the resulting runtime behaviour are non-trivial issues, and tool support is needed to make such approaches practical.

Tools Involved The main focus of this tool chain lies on testing of dynamic service discovery, hence the most important tool is the Dino Broker used for service discovery. Serving input to Dino is the Modes Parser and Browser Tool which handles translation of modes from the UML2 models. Dino also requires WSDL and OWL documents for service specification which can, in part, be generated by the VIATRA2 SOA2WSDL transformation tool. Again, the initial mode specification is done in UML2, for which a UML2 modeler is required.

Data Flow The chain starts with the user who employs a UML modeler to design a model of a SOA system enhanced with mode specifications and the required constraints on services. The Modes Parser and Browser Tool is then used to convert these specifications to input for the Dino Broker. In parallel, the services to be discovered are deployed to the Dino runtime, either from pre-existing OWL/WSDL specifications or from those generated by the SOA2WSDL transformation. Finally, the developer can employ the Dino Broker frontend which is available through the SDE to test-drive the service discovery, and once satisfied, use the generated documents for the final implementation.

Results The ability to generate input for Dino from UML2 and test-driving the discovery right from within the development environment greatly speeds up the process of finding the right mode and constraint specifications. Automation allows writing test cases for the complete process, thus the user may change the specifications at the beginning of the chain and verify the output stemming from an actual discovery run with the Dino Broker, thus saving time and effort in debugging.

6 Conclusions and Future Work

In this chapter, we have discussed the need for, requirements of, implementation, and usage of a tool integration platform for the development of service-oriented software systems, the SENSORIA Development Environment (SDE). Based on a service-oriented architecture itself, the SDE contains tools for modelling and analysing service artefacts as well as generating code and supporting services at runtime, allows remote invocation of tool functionality, and enables composition of tools by a textual and graphical orchestration mechanism.

Furthermore, we have discussed integrated tools providing support for the development of SOA software, and have outlined how to use tools in combination on top of the SDE.

We believe that thinking of individual development tools as services and including SOA features like self-describing services, remote invocation, and orchestration into our tooling environment greatly extends the applicability of the integrated tools. By including transformation tools, we ensure that using analysis tools is possible without understanding the details of the underlying formal specifications, thus allowing more developers to profit from rigourous verification of their systems.

The SDE, including the integrated tools, is available for download at our dedicated tooling website, `http://svn.pst.ifi.lmu.de/trac/sct`. The website also contains a tutorial for tool integration and videos demonstrating the SDE in action.

References

[Ecl09a] Eclipse Foundation. Eclipse Equinox - Implementation of the OSGi R4 core framework specification. `http://www.eclipse.org/`, 2009.

[Ecl09b] Eclipse Foundation. The Eclipse Open Source Community and Java IDE. `http://eclipse.org/equinox/`, 2009.

[Erl05] Thomas Erl. *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall International, 2005.

[MRH08] Philip Mayer, István Ráth, and Ádám Horváth. Report on the Sensoria Development Environment - Second Version. Technical report, LMU München, 2008.

[MSK08] Philip Mayer, Andreas Schroeder, and Nora Koch. MDD4SOA: Model-Driven Service Orchestration. In *The 12th IEEE International EDOC Conference (EDOC 2008)*, pages 203–212, Munich, Germany, 2008. IEEE Computer Society.

[OSG08] OSGi Alliance. Osgi specification release 4. `http://www.osgi.org/Specifications/`, 03 2008.

[RAR07] Jan S. Rellermeyer, Gustavo Alonso, and Timothy Roscoe. R-OSGi: distributed applications through software modularization. In *Middleware '07: Proceedings of the*

ACM/IFIP/USENIX 2007 International Conference on Middleware, pages 1–20, New York, NY, USA, 2007. Springer-Verlag New York, Inc.

[WBC$^+$09] Martin Wirsing, Laura Bocchi, Alan Clark, Jose Fiadeiro, Stephen Gilmore, Matthias Hölzl, Nora Koch, Philip Mayer, Rosario Pugliese, and Andreas Schroeder. Sensoria: Engineering for service-oriented overlay computers. In Elisabetta Di Nitto, Anne-Marie Sassen, Paolo Traverso, and Arian Zwegers, editors, *At Your Service: Service-Oriented Computing from an EU Perspective*, pages 159–182. MIT Press, 2009.

[WCL$^+$05] Sanjiva Weerawarana, Francisco Curbera, Frank Leymann, Tony Storey, and Donald F. Ferguson. *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging and More*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.