Sensoria

016004

Software Engineering for Service-Oriented Overlay Computers

www.sensoria-ist.eu

D7.4.c

Report on the Sensoria Development Environment

Second Version

Lead contractor for deliverable: LMU
Author(s): Philip Mayer(LMU), István Ráth (BUTE), Ádám Horváth (BUTE)

Due date of deliverable: August 31, 2008
Actual submission date: August 31, 2008
Revision: Final (3)
Classification: 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

This document contains the second report on the Sensoria Development Environment (SDE) (formerly known as Sensoria CASE tool, SCT), which supports the development of service-oriented software by integrating the various tools developed as part of the Sensoria project as well as useful external tools. The integrated tools span the complete lifecycle of software development from modelling tools over transformation and analysis tools to deployment and runtime.

This document discusses the SDE version 4, which was developed in the past reporting period. After a general overview over the basic ideas and implementation of the SDE, we will introduce newly added features, list all integrated tools, and provide example workflows created with the help of the SDE.

Contents

1 Introduction

Within the course of the Sensoria project [1], a variety of tools are being developed which aid developers in using the newly created foundational approach to developing service-oriented software. These tools range from modelling over analysis and transformation to deployment and runtime tools.

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 lifecycle and in many cases deliver artefacts which may serve as input to other tools. Thus, developers should be able to use these tools in combination to automate as many tasks as possible.

Consequently, an integration approach was devised which allows tool developers to easily add their tools to an integrating platform, called the Sensoria Development Environment (SDE) [2], formerly known as the Sensoria CASE Tool [3]. The SDE enables users of the tool to discover and use the tools developed within Sensoria. It is based on a Service-Oriented Architecture (SOA) approach, where each tool is represented as a service.

The SDE is under development since the very beginning of Sensoria and has already been described in two previous deliverables. The first of these, D7.4.a [3], listed general requirements for an integration tool and has shown a first list of tools to be integrated. In the second deliverable, D7.4.b [4], we have reported on our implementation of the SDE and the tools which have been integrated.

In this current deliverable, we present the SDE Version 4, which was developed in the past reporting period. As we will show, the tool now contains all necessary functionality for reaching the envisioned goals within Sensoria.

This deliverable is structured as follows: In Chapter 2, we will revisit the principles on which the Sensoria Development Environment is built, along with the general functionality. Chapter 3 will then detail new functionality implemented since D7.4.b. In Chapter 4, we will list integrated tools, and use them in Chapter 5 to show some sample integration workflows. We conclude in Chapter 6.

2 The Sensoria Development Environment

The Sensoria Development Environment (SDE) is an automated development environment for service artefacts, which offers, through integrated tools, service modelling, analysis, code generation, and runtime functionality. The aim of the SDE is to provide the various tools required for developing services, including formal analysis tools, in one consistent and integrated environment, offering state-of-the-art research techniques in an easy-to-use fashion to developers. This is achieved through the following core features in the platform:

- *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 analysis tools hosted in the SDE are presented as discoverable, technology independent 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 tools.
- *A Focus On Usability* – to allow developers to use formal tools without requiring them to understand the underlying formal semantics, the SDE employs automated model transformations which translate between high-level models and formal specifications, thus closing the gap between those two worlds.

The SDE is built on the Eclipse platform [5] and its underlying service-oriented OSGi [6] framework. Fundamentally, tools are integrated as services; specifically, in this case, as OSGi bundles. To enable access to integrated tools, the SDE contains a discovery service in the spirit of a UDDI registry. Upon installation, a tool is registered with its offered functionality in the form of public API functions which provide UI-independent access to its core functionality. The functions offered by each tool are made available for querying and execution by the SDE core, which enables both manual invocations and automation.

Section 2.1 details how the tools integrated into the SDE can be used in an overall service engineering approach. Section 2.2 will give an overview on automating tools, and section 2.3 on developer access.

2.1 Using the SDE

Through various integrated tools, the SDE offers functionality which covers the complete model-driven process of service engineering, which is shown in Fig. 1. After starting out 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.

Currently, nearly twenty tools have been integrated into the SDE. Most of the tools fall into the following three major categories:

- *Modelling Functionality* – this includes graphical editors for familiar modelling languages such as UML, which allow for intuitive modelling on a high level of abstraction, and also text- and tree-based editors for formal languages like process calculi.
- *Model Transformation Functionality, including Code Generation* – the SDE offers automated model transformation 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) is supported.
- *Formal Analysis Functionality* – the SDE offers model checking and numerical solvers for stochastic methods based on process calculi code defined by the user or generated by model transformation.

Chapter 4 will list integrated tools for all of these categories.

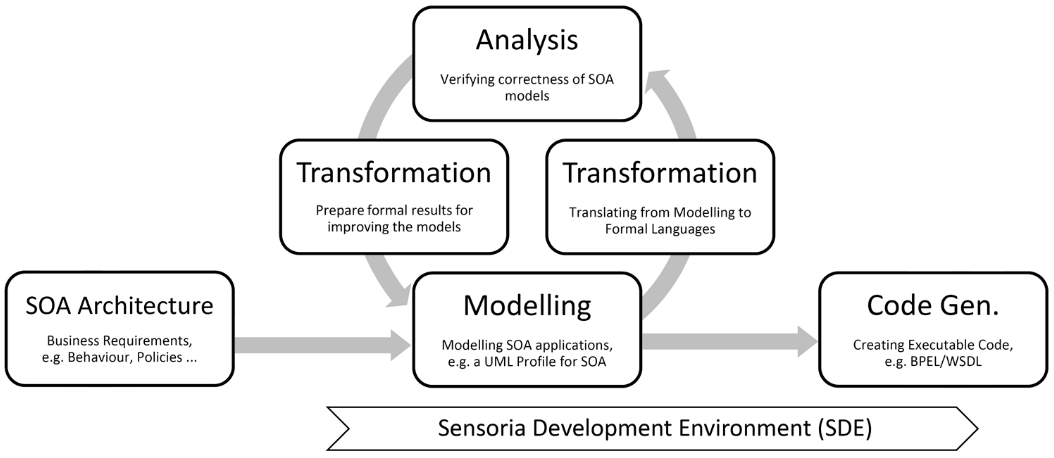
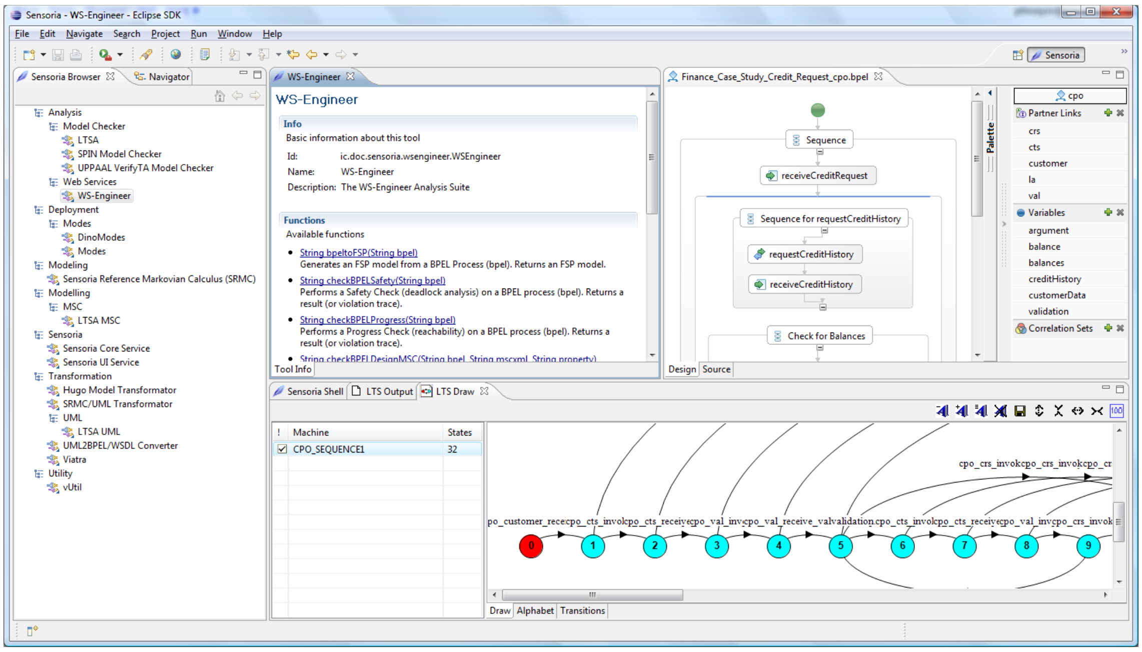

Analysis
Verifying correctness of SOA models
Transformation
Prepare formal results for improving the models
Transformation
Translating from Modelling to Formal Languages
SOA Architecture
Business Requirements, e.g. Behaviour, Policies ...
Modelling
Modelling SOA applications, e.g. a UML Profile for SOA
Code Gen.
Creating Executable Code, e.g. BPEL/WSDL
Sensoria Development Environment (SDE)

Fig. 1: Model-Driven Development Approach as used in the SDE

As a general overview, Fig. 2 shows a screenshot of the SDE inside Eclipse. On the left hand side, the tool browser shows installed tools available for invocation and automation. In the middle, an integrated tool for qualitative analysis (WS-Engineer) is shown in more detail with its available functionality. On the right and bottom, additional views show a BPEL process using the Eclipse BPEL tools, and output of the WS-Engineer tool.

In general, each tool available through the SDE is listed in the tool browser on the left hand side. Double-clicking a tool yields more information about the tool and its functionality, as shown for the WS-Engineer example. Each function can be invoked by clicking the link and providing the parameters; in-between function calls, a blackboard stores the data. However, most tools also provide their own customized user interface which can be invoked as described in their documentation.



WS-Engineer

Info

Basic information about this tool

Id: ic.doc.sensoria.wsengineer.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)

!	Machine	States
☑	CPO_SEQUENCE1	32

Fig. 2: SDE: Tool Browser (left), Integrated Tool "WS-Engineer" (middle), BPEL Analysis Example (right/bottom)

2.2 Automating Tasks with the SDE

During software development and analysis of software systems, it is often desirable to run several analyses as a suite, perhaps passing input from one tool to the other, and gathering and presenting the output in a single place – in other words, orchestrating tools to perform as a whole. The SDE offers the ability to create such orchestrations by several means. First, there is a graphical orchestration editor which employs UML activity diagrams for modelling data flow between various tools. A second option is to use the more low level scripting language JavaScript. The compositions invoke the APIs provided by the installed tools, thus in effect creating a new service which orchestrates the functions of the referenced tools.

The ability to orchestrate tools is of particular importance in the context of formal analysis tools. In most cases, these tools require input specified in a formal language, and provide their output on this level as well. As discussed before, service engineers might not be willing to invest the time to handle such artefacts. Instead, the tools should accept input on the modelling level of the service engineer, and report back on the same level. One way of achieving this is through automated model transformations which handle the translation in-between these levels. Chapter 5 will show some examples of such automation.

2.3 Adding and Managing Tools

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 new 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 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 (semantic) 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.

3 New Features and Functionality

Version 4 of the Sensoria Development Environment contains new functionality which partly stems from the requirements identified in the last deliverable and partly from features newly identified during the current reporting period. In this chapter, the main new functionality will be detailed, along with some information about smaller additional changes.

3.1 Composition Infrastructure

Version 3 of the SDE introduced the ability to write JavaScript to compose several tools, in effect creating an executable program orchestrating several tools into one common workflow. However, using such workflows as new tools themselves was not yet possible. In version 4, this ability has been added, along with an additional method for creating orchestrations, namely through a graphical, UML activity diagram-like editor.

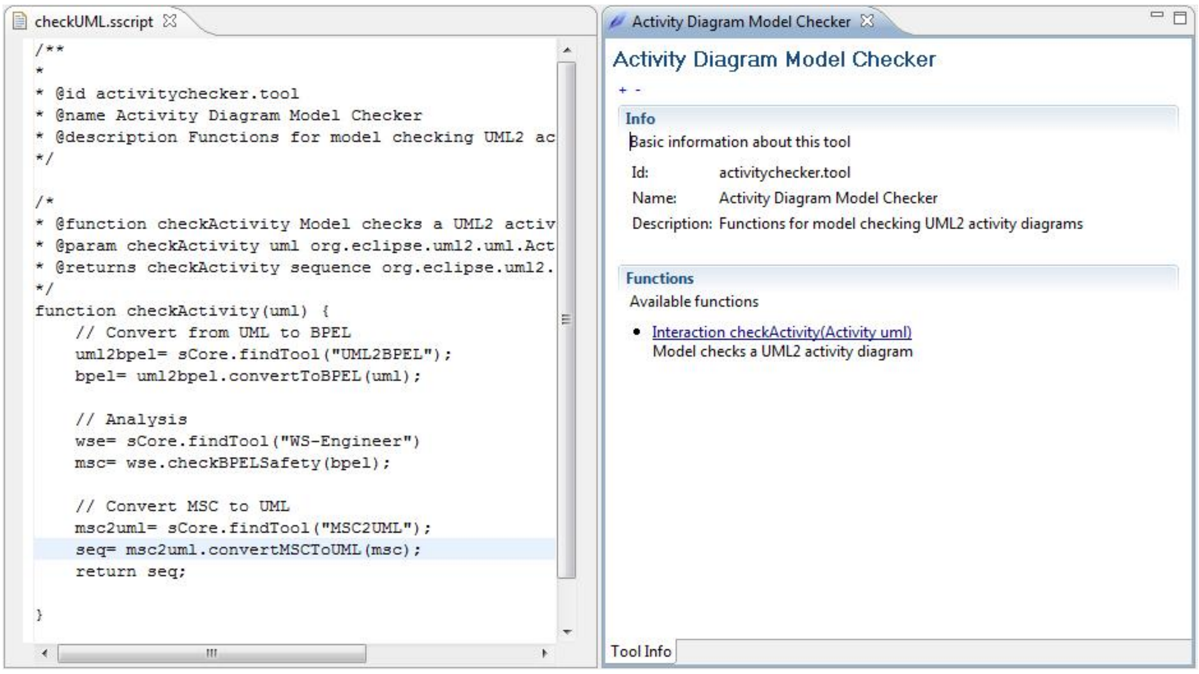
3.1.1 Orchestrating with JavaScript

The ability to use tool API directly within JavaScript enables JavaScript writers 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.



```
/**
*
* @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

Fig. 3: JavaScript orchestration and generated tool

As an example, Fig. 3 (left) shows a script for converting UML2 activity diagrams to BPEL, then analyzing them using the WS-Engineer tool, and finally converting the result back to UML2 sequence diagrams showing the call path in violation. Fig. 3 (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.

3.1.2 Graphical Orchestration

Besides the ability to use JavaScript for orchestration as indicated above, the SDE now 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. 4, which shows the orchestration introduced in the previous section as a graphical diagram, 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.

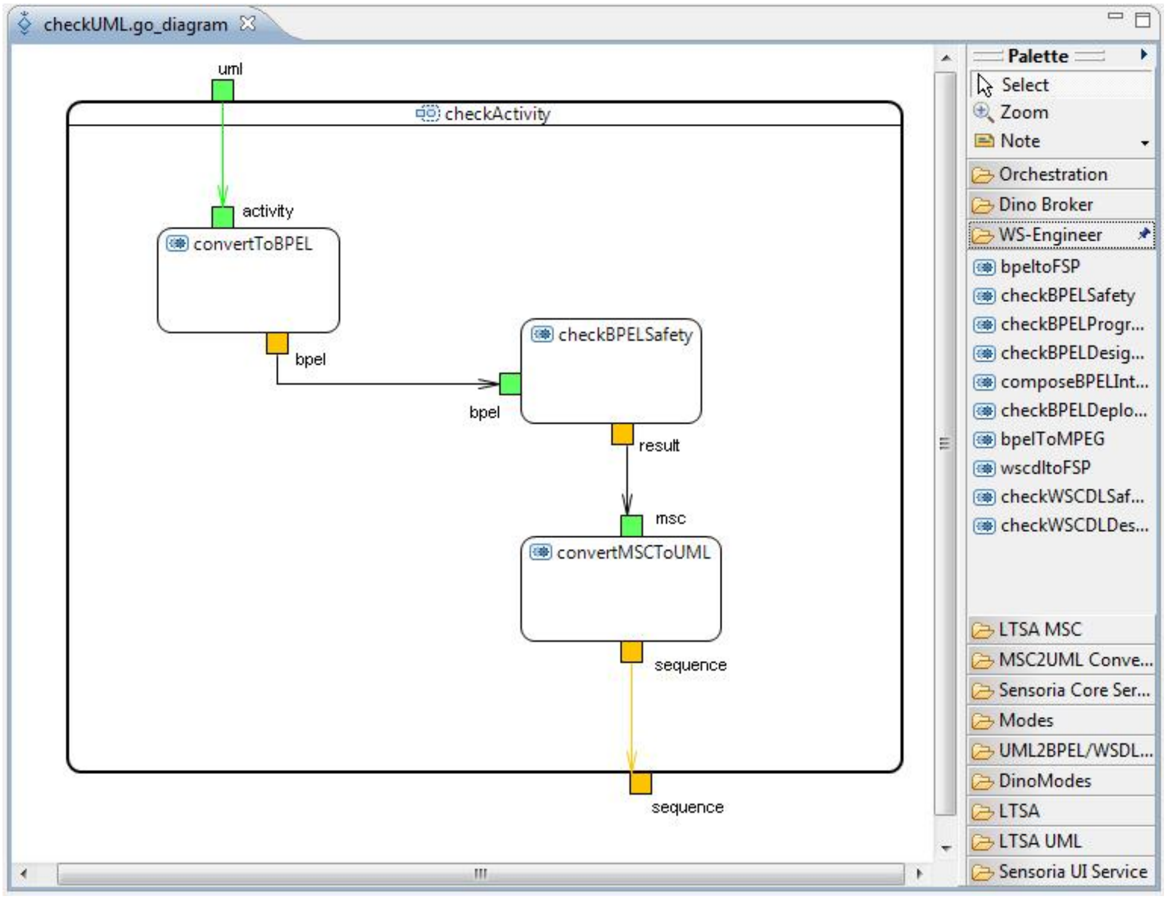


Fig. 4: Graphical Orchestration

3.2 Distributed Multi-Core Operation

As a layer on top of Eclipse, the Sensoria Development Environment is a client-side development environment without any connections to other installed versions or servers per se. Although it is possible, and in fact has been implemented, to access Web services or other software components on other servers by wrapping the calls in a tool bundle, this process was not yet natively available within the SDE core.

Therefore, version 4 of the SDE has been extended with a peer-2-peer communication model which enables several installations of the SDE to connect to each other via a TCP/IP network. This extension enables users of the SDE to transfer long-running analyses or transformations to other installations of the SDE which might run on a more powerful machine. Once a connection to a remote SDE installation has been made, calls to remote tools and functions are handled transparently.

To enable this functionality, the SDE has been extended in various ways. First of all, an additional layer beneath the core has been introduced to enable the SDE to deal with multiple cores, some local, and some remote (note that one local core is always present, thus the implementation is fully backwards-compatible). Secondly, the SDE API and basic UI are now aware of multiple cores and are therefore able to display tools as well as blackboard data from remote cores. Also, the JavaScript shell has been extended with new functionality for handling additional cores and their tools.

Finally, tools and their functionality in the SDE environment are now bound to change more often during the lifetime of an SDE instance due to dynamic registering of remote SDE cores. The SDE environment has thus been extended for runtime reloading of installed and available tools.

3.3 Distributed Updating

The Sensoria tools are created, maintained, and made available for download at the partner sites where they are developed. The prevailing mechanism for doing so is using Eclipse update sites which enable users to simply point their Eclipse instance to a particular URL and thus initiate the download- and installation process.

However, the user first needs to discover which update sites are available and which tools and functions they offer. Therefore, we have created a web-based platform in which tools and their update sites as well as scripts can be registered and viewed. A user then may select one or several tools and get a listing of the necessary update sites which can be readily imported into Eclipse and used to download and install the tools.

This approach leaves the responsibility for each tool, including updates, at each individual partner which ensures timeliness of updates, but at the same time offers users with a convenient online interface to discover and then install the tools they are interested in.

3.4 Inline Installation

As a tool bundle can only be implemented in Java, some of the bundles are implemented by wrapping existing functionality written in native languages. Thus, this native code needs to be present in the system somehow to be accessed by the wrapper.

The SDE provides a hook for tools which need to extract, or download, additional code or binaries upon installation. Tools may implement a newly created interface, `ISensoriaInstallableTool`, which provides both an `install` and an `uninstall` routine. The first is called by the SDE after the tool has been started for the first time, whereas the latter may be invoked by the user to uninstall the additional elements from the system.

3.5 Additional Changes

In addition to the larger features identified above, the Sensoria Development Environment has also seen other improvements such as bug fixes or minor feature additions. Among them are:

- *Data Descriptions:* Parameters and return types of functions of tools now have their own descriptions. This allows the user interface to show more information to the user.
- *Model Information:* Tools may now provide additional support for the model classes they use, or require, in their API. This allows the SDE to react to each model element individually, for example by opening an Eclipse editor or showing a specialized dialog. In particular, the blackboard features more actions (like “Open in Editor”) and the ability to display object contents.
- *Array/Collection awareness:* The invocation wizard is able to deal with Arrays and Collections as inputs.
- *More functionality for scripts:* The SDE core now comes with two preinstalled tools which contain several meta-functions required for data handling and tool configuration. The tools include functions for working with files in and out of the workspace, handling tool options, and opening model elements in the Eclipse editors.

Information about additional changes can be found in the bug tracker on the SDE web site [2].

4 Tool Integrations

This chapter lists all tools which have been integrated into the SDE platform, sorted by integrated category. An integrated tool offers its functionality in the SDE platform through a tool registration and API which can be invoked either manually be the user or via the orchestration mechanism. Note that most of the tools additionally provide their own UI into the Eclipse platform for easier access.

4.1 Modelling

4.1.1.1 ArgoUML

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

URL: http://argouml.tigris.org/

4.1.1.2 Rational Software Architect

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

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

4.1.1.3 MagicDraw

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

URL: http://www.magicdraw.com/

4.2 Transformation and Deployment

4.2.1.1 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.

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

4.2.1.2 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.

URL: http://dev.eclipse.org/viewcvs/indextech.cgi/gmt-home/subprojects/VIATRA2/index.html

4.2.1.3 SOA2WSDL-Transformation

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

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

4.2.1.4 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.

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

4.2.1.5 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.

URL: http://www.mdd4soa.eu/

4.2.1.6 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.

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

4.2.1.7 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.

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

4.2.1.8 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.

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

4.3 Analysis

4.3.1.1 LTSA

LTSA is a verification tool for concurrent systems. It mechanically 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.

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

4.3.1.2 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.

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

4.3.1.3 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.

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

4.3.1.4 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.

URL: http://spinroot.com/

4.3.1.5 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.).

URL: http://www.uppaal.com/

4.3.1.6 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.

URL: http://www.mdd4soa.eu/

4.3.1.7 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.

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

4.4 Runtime

4.4.1.1 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.

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

5 Tool Chaining

The tools listed in the previous chapter can be combined in various ways to achieve different transformations and analyses. Fig. 5 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. The tool chains may be realized 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 customized 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.

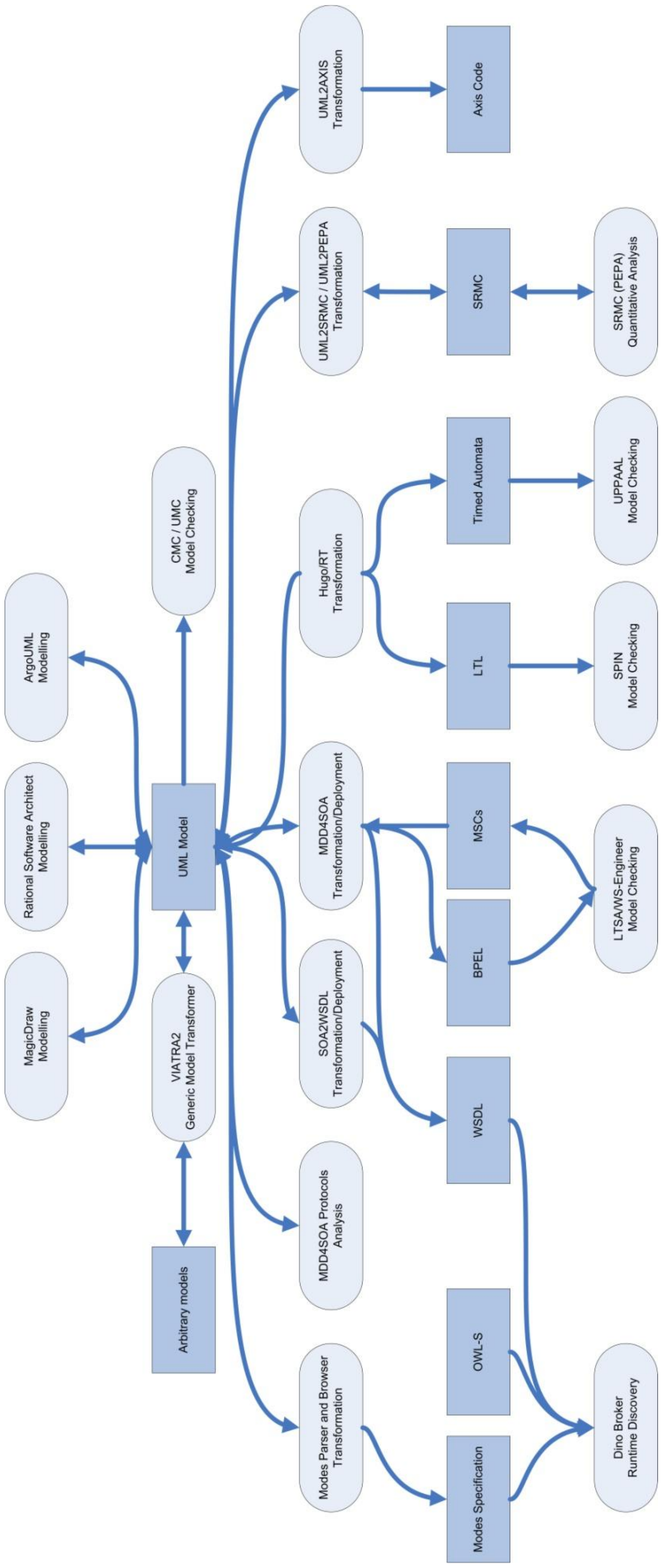

MagicDraw Modelling
Rational Software Architect Modelling
ArgoUML Modelling
Arbitrary models
VIATRA2 Generic Model Transformer
UML Model
CMC / UMC Model Checking
Modes Parser and Browser Transformation
MDD4SOA Protocols Analysis
SOA2WSDL Transformation/Deployment
MDD4SOA Transformation/Deployment
Hugo/RT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

Fig. 5: Tool Chains in the SDE

Tools Involved

This tool chain includes a UML modeller 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 modeller 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 Analyzer, which either reports no protocol violations or creates a violation trace, again as an UML2 diagram. 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 interaction of such threads needs to be analyzed in order to ensure that no undesirable effects (such as deadlocks) occur. Furthermore, it is not always clear how the system’s 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 modellers 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 modeller 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 instigate 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 analyzers. 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 – of 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 modeller is required.

Data Flow

The chain starts with the user who employs a UML modeller 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 Related Publications

Due to the integrative nature of the SDE, this deliverable contains various references to other work done as part of Sensoria. First of all, the SDE itself has already been described in two deliverables: In [3], we have listed initial requirements for the SDE, followed by a description of the first spikes and development efforts in [4].

The SDE is also described in several publications, mostly in combination with integrated tools. In [7], we introduce the overall Sensoria approach including the SDE. The benefits of an open tool integration platform for service analysis is detailed in [8]. Combining multiple tools by means of scripting is covered in [9]. Finally, [10] describes the use of the SDE in an overall model driven development process.

Most of the tools listed in this deliverable are described in more detail in their own respective deliverables. The following list gives an overview of these, sorted by SDE category.

- *Modelling*: The UML4SOA profile used as input for many of the Sensoria tools is described in D1.4.a [11] and in Th0.4.b [12].
- *Analysis*: Qualitative and quantitative analysis methods and tools are part of WPs 3 and 4, with additional work done in other work packages and well. An overview is given in Th0.3.a [13]. In particular, the CMC/UMC tools are described in D3.d [14].
- *Transformations*: The MDD4SOA transformers, which include the UML2BPEL/WSDL, UML2Jolie, and UML2Java transformations, as well as the deployment transformations to the Apache Platform are

described in D6.4.b [15]. Further advances on model transformations (e.g. event-driven transformations) and transformations for performability analysis are described in Th0.4.b [12].

- *Deployment*: Deployment support for dynamic and adaptive service compositions are described in D6.1.c [16]. The Modes Tool and Dino Broker are described in D6.3.c [17].

Finally, the SDE has also been described in relation to the case studies in D8.7 [18].

7 Future Work

In the following months, we will continue to work with the Sensoria partners to ensure that the platform works as intended and to help in integrating possible additional tools. We will also investigate a concept for the future of the SDE within the open source community. As before, we will also use demonstrator instances of the SDE to present our work to a broader audience; currently planned is a visit to CeBIT Australia in 2009.

8 Summary

In this deliverable, we have described the work on the Sensoria Development Environment in the reporting period M24 to M36. As laid down in the technical annex, the SDE was intended to be finished during this time span and we are happy to report that this goal has been met.

The Sensoria Development Environment is now in its fourth version. With nearly twenty integrated tools, the ability to work across machines, and a graphical orchestration mechanism which yields new tools as a result of the orchestration, the SDE is a fairly complete environment for using the Sensoria tools for developing service-oriented software systems. As it is based on Eclipse, it may be readily integrated with other tools based on this platform.

9 Bibliography

[1] Sensoria Team, "Sensoria: Service Engineering for Service-Oriented Overlay Computers", http://www.sensoria-ist.eu/, [2005-2009]. Last visited: 24 July 2008.

[2] "The Sensoria Development Environment (SDE)", http://svn.pst.ifi.lmu.de/trac/sct, [2006-2008]. Last visited: 24 July 2008.

[3] H. Baumeister, *D7.4a: CASE Tool: Initial Requirements*, Deliverable of EU Project Sensoria, 2006.

[4] P. Mayer, and H. Baumeister, *D7.4b: Report on the Sensoria CASE Tool: Description and Evaluation*, Deliverable of EU Project Sensoria, 2007.

[5] Eclipse Foundation, "Eclipse - an open development platform", http://www.eclipse.org/, [2008]. Last visited: 24 July 2007.

[6] OSGi Alliance. "OSGi Specification Release 4", March 19, 2008; http://www.osgi.org/Specifications/.

[7] M. Wirsing, L. Bocchi, A. Clark, J. L. Fiadeiro, S. Gilmore, M. Hölzl, N. Koch, P. Mayer, R. Pugliese, and A. Schroeder, *Sensoria: Engineering for Service-Oriented Overlay Computers in At your service: Service Engineering in the Information Society Technologies Program*: MIT Press, 2008.

[8] H. Foster, and P. Mayer, "Leveraging Integrated Tools for Model-Based Analysis of Service Compositions". *Third Intl. Conference on Internet and Web Applications and Services (ICIW 2008) Athens, Greece*, 2008.

[9] A. Argent-Katwala, A. Clark, H. Foster, S. Gilmore, P. Mayer, and M. Tribastone, "Safety and Response-Time Analysis of an Automotive Accident Assistance Service". *Proceedings of 3rd International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA 2008), Porto Sani, Greece*.

[10] M. Wirsing, M. Hölzl, N. Koch, P. Mayer, and A. Schroeder, "Service Engineering: The Sensoria Model Driven Approach". *Proceedings of Software Engineering Research, Management and Applications (SERA 2008), Prague, Czech Republic*.

[11] N. Koch, P. Mayer, R. Heckel, L. Gönczy, and C. Montangero, "D1.4a: UML for Service-Oriented Systems", Deliverable of EU project Sensoria, Reporting Period 2007, 2007.

[12] D. Varró, *Th0.4.b: Methodologies for MDA and Deployment (second version)*, Deliverable of EU project Sensoria, Reporting Period 2008, 2008.

[13] P. Quaglia, and S. Gilmore, *Th0.3.a: Quantitative and qualitative measurements of quality of service and service level agreements*, Deliverable of EU project Sensoria, Reporting Period 2008, 2008.

[14] F. Nielson, and H. Pilegaard, *D3.d: Tools and Verification*, Deliverable of EU project Sensoria, Reporting Period 2008, 2008.

[15] L. Gonczy, *D6.4.b: Model-Driven Transformations for Deployment - Second Version*, Deliverable of EU project Sensoria, Reporting Period 2008, 2008.

[16] D. Rosenblum, *D6.1.c: Deployment Support for Dynamic and Adaptive Service Composition*, Deliverable of EU project Sensoria, Reporting Period 2008, 2008.

[17] D. Rosenblum, *D6.3.c: Enhanced service-oriented deployment platform (Prototype)*, Deliverable of EU project Sensoria, Reporting Period 2008, 2008.

[18] N. Koch, and H. Foster, *D8.7: Relations among Case Studies and Theme 3 Results*, Deliverable of EU project Sensoria, Reporting Period 2008, 2008.