Network of Excellence

Deliverable D2.3

Second release of the SDE for Security-Related Tools
<table>
<thead>
<tr>
<th>Deliverable Number</th>
<th>D2.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title of Deliverable</td>
<td>Second release of the SDE for Security-Related Tools</td>
</tr>
<tr>
<td>Nature of Deliverable</td>
<td>R</td>
</tr>
<tr>
<td>Dissemination Level</td>
<td>Public</td>
</tr>
<tr>
<td>Internal Version Number</td>
<td>3.0</td>
</tr>
<tr>
<td>Contractual Delivery Date</td>
<td>September 30 2012</td>
</tr>
<tr>
<td>Actual Delivery Date</td>
<td>October 15 2012</td>
</tr>
<tr>
<td>Contributing WPs</td>
<td>WP2</td>
</tr>
</tbody>
</table>
| Editor(s) | Marianne Busch (LMU)  
Nora Koch (LMU) |
| Author(s) | Kristian Beckers (UDE)  
Marianne Busch (LMU)  
Vincenzo Ciancia (CNR)  
Manuel Clavel (IMDEA)  
Said Daoudagh (CNR)  
Miguel Ángel García de Dios (IMDEA)  
Bart Jacobs (KUL)  
Nora Koch (LMU)  
Francesca Lonetti (CNR)  
Martín Ochoa (SIEMENS)  
Cristo Reyes (ATOS)  
Fredrik Seehusen (SINTEF)  
Mathieu Turuani (INRIA)  
Bjørnar Solhaug (SINTEF)  
Koen Yskout (KUL) |
| Reviewer(s) | Bendavid Olivier (INRIA)  
Manuel Clavel (IMDEA)  
Marinella Petrocchi (CNR) |
Abstract

This report includes a description of the extensions implemented in the Service Development Environment (SDE) for security-related tools. We also report on new tools integrated into the platform, i.e. UWE2XACML, X-CREATE, XACML2FACPL, AbsInt, and PRRS. Furthermore, examples of how to use integrated SDE tools are presented. The main aim of extending the number of integrated tools is to be able to create tool chains so that several tools can be used in a row. We introduce a tool chain example that transforms UWE access control models to FACPL, which is done via the XACML format.

Keyword List

Service Development Environment, Integrated Development Environment, Tool Integration, Tool Workbench
# Document History

<table>
<thead>
<tr>
<th>Version</th>
<th>Type of Change</th>
<th>Author(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>Table of Contents (alpha)</td>
<td>Marianne Busch (LMU)</td>
</tr>
<tr>
<td>0.2</td>
<td>Introduction</td>
<td>Nora Koch (LMU)</td>
</tr>
<tr>
<td>0.3</td>
<td>CORAS usage example</td>
<td>Fredrik Seehusen (SINTEF) and Bjørnar Solhaug (SINTEF)</td>
</tr>
<tr>
<td>0.4</td>
<td>SDE extension, MagicUWE example, tool chains, integration of UWE2XACML and XACML2FACPL in SDE</td>
<td>Marianne Busch (LMU)</td>
</tr>
<tr>
<td>0.6</td>
<td>AbsInt integration</td>
<td>Martin Ochoa (SIEMENS) and Marianne Busch (LMU)</td>
</tr>
<tr>
<td>0.7</td>
<td>UML4PF usage example</td>
<td>Kristian Beckers (UDE)</td>
</tr>
<tr>
<td>0.8</td>
<td>EOS usage example</td>
<td>Manuel Clavel (IMDEA) and Miguel Ángel García de Dios (IMDEA)</td>
</tr>
<tr>
<td>0.9</td>
<td>CL-atse and Orchestrator</td>
<td>Mathieu Turuani (INRIA)</td>
</tr>
<tr>
<td>1.0</td>
<td>PRRS integration</td>
<td>Marina Soledad Egea González (ATOS) and Cristo Reyes (ATOS)</td>
</tr>
<tr>
<td>1.1</td>
<td>Description of UWE2XACML and XACML2FACPL, summary and outlook</td>
<td>Nora Koch (LMU)</td>
</tr>
<tr>
<td>1.2</td>
<td>HospInfo example, corrections, formatting, bibliography and footnotes</td>
<td>Marianne Busch (LMU)</td>
</tr>
<tr>
<td>1.3</td>
<td>Tool chain example, structure of examples, uniform layout and style, corrections</td>
<td>Nora Koch (LMU)</td>
</tr>
<tr>
<td>1.4</td>
<td>Jalapa example</td>
<td>Vincenzo Ciancia (CNR)</td>
</tr>
<tr>
<td>1.5</td>
<td>References and images</td>
<td>Marianne Busch (LMU)</td>
</tr>
<tr>
<td>2.0</td>
<td>Improvements according to internal reviews</td>
<td>LMU and others</td>
</tr>
<tr>
<td>3.0</td>
<td>Approved by the internal reviewers</td>
<td>Bendavid Olivier (INRIA), Manuel Clavel (IMDEA) and Marinella Petrocchi (CNR)</td>
</tr>
</tbody>
</table>
Table of Contents

LIST OF FIGURES ........................................................................................................ 11
LIST OF TABLES ........................................................................................................... 13
1 INTRODUCTION ........................................................................................................ 15

2 EXTENSIONS OF THE SDE ...................................................................................... 17
  2.1 Bugfixes and Updates ......................................................................................... 17
  2.2 CBK Search ........................................................................................................ 18
  2.3 Managing Examples for Integrated Tools ......................................................... 18

3 FURTHER INTEGRATED TOOLS ............................................................................ 21
  3.1 UWE2XACML ...................................................................................................... 22
    3.1.1 Features ....................................................................................................... 22
    3.1.2 Integration into the SDE .............................................................................. 23
  3.2 X-CREATE .......................................................................................................... 23
    3.2.1 Features ....................................................................................................... 23
    3.2.2 Integration into the SDE .............................................................................. 25
  3.3 XACML2FACPL .................................................................................................. 25
    3.3.1 Features ....................................................................................................... 26
    3.3.2 Integration into the SDE .............................................................................. 27
  3.4 AbsInt .................................................................................................................. 27
    3.4.1 Features ....................................................................................................... 28
    3.4.2 Integration into the SDE .............................................................................. 29
  3.5 PRRS .................................................................................................................. 30
    3.5.1 Features ....................................................................................................... 30
    3.5.2 Integration into the SDE .............................................................................. 32

4 USAGE EXAMPLES OF INTEGRATED TOOLS ............................................................ 33
  4.1 Avantssar-atse (CL-ATSE) ................................................................................. 33
    4.1.1 Features ....................................................................................................... 33
    4.1.2 Working with CL-Atse in SDE ..................................................................... 34
  4.2 Avantssar Orchestrator ....................................................................................... 34
    4.2.1 Features ....................................................................................................... 36
    4.2.2 Working with the Orchestrator in SDE ......................................................... 36
  4.3 CORAS Tool ....................................................................................................... 38
    4.3.1 Features ....................................................................................................... 38
    4.3.2 Working with the CORAS Tool in SDE ......................................................... 38
  4.4 EOS (Eye OCL Software) ..................................................................................... 40
    4.4.1 Features ....................................................................................................... 40
    4.4.2 Working with EOS in SDE ......................................................................... 40
  4.5 Jalapa ................................................................................................................... 42
    4.5.1 Features ....................................................................................................... 42
    4.5.2 Working with Jalapa in SDE ....................................................................... 43
List of Acronyms

AES  Advanced Encryption Standard
API  Application Programming Interface
ASCENS  Autonomic Service-Component Ensembles
BNF  Backus-Naur-Form
CASE  Computer-Aided Software Engineering
CBK  Common Body of Knowledge
EHR  Electronic Health Record
FACPL  Formal Access Control Policy Language
FI  Future Internet
FSA  Finite State Automata
GUI  Graphical User Interface
HIS  Hospital Information System
IDE  Integrated Development Environment
LOP  Loan Origination Process
LTL  Linear Temporal Logic
NESSoS  Network of Excellence on Engineering Secure Future Internet Software Services and Systems
OCL  Object Constraint Language
SENSORIA  Software Engineering for Service-Oriented Overlay Computers
SDE  Service Development Environment
SDLC  Software Development Lifecycle
SOA  Service-Oriented Architecture
UML  Unified Modeling Language
UML4PF  UML for Problem Frames
URL  Uniform Resource Locator
UWE  UML-based Web Engineering
XACML  eXtensible Access Control Markup Language
XMI  XML Metadata Interchange
XML  Extensible Markup Language
List of Figures

Figure 2.1: Improved Error Handling ................................................................. 17
Figure 2.2: CBK interface within the SDE .......................................................... 18
Figure 2.3: Executing a method of a tool manually ............................................. 19
Figure 2.4: Loading a SDE example set .............................................................. 19
Figure 3.1: Overview of integrated tools ............................................................ 21
Figure 3.2: UWE2XACML result in SDE .............................................................. 22
Figure 3.3: X-CREATE requests derivation ......................................................... 24
Figure 3.4: X-CREATE as a SDE component ....................................................... 25
Figure 3.5: XACML2FACPL result in SDE .......................................................... 26
Figure 3.6: Overview of the AbsInt SDE Plugin .................................................... 28
Figure 3.7: Partial abstract cache state ............................................................... 29
Figure 3.8: PRRS use cases .................................................................................. 30
Figure 3.9: PRRS solutions folder ....................................................................... 31
Figure 3.10: PRRS Add Artifact ......................................................................... 31
Figure 3.11: PRRS Add Class ............................................................................ 31
Figure 4.1: Aslan GUI’s main page (defining the Environment) .......................... 35
Figure 4.2: Improved Avantssar’s LOP case-study ............................................. 37
Figure 4.3: Screenshot of the CORAS tool ......................................................... 39
Figure 4.4: Validating with EOS/SDE the ActionGUI security model for the EHealth Case Study ... 41
Figure 4.5: Jalapa policy editor ......................................................................... 42
Figure 4.6: Jalapa SDE integration .................................................................... 44
Figure 4.7: Examples for the Encrypt function ................................................. 45
Figure 4.8: UML4PF Tool Realization Overview .............................................. 46
Figure 4.9: UML4PF: Screenshot of Problem Diagram for R01 ....................... 48
Figure 4.10: OCL Validator inside the SDE platform ......................................... 48
Figure 4.11: loadUML4PFModel parameter ................................................................. 49
Figure 4.12: ModelGenerator working ................................................................. 50
Figure 4.13: Model Validation in progress ............................................................. 50
Figure 4.14: X-CREATE project ............................................................... 53
Figure 4.15: X-CREATE results ................................................................. 53

Figure 5.1: UWE2FACPL tool chain ................................................................. 56
Figure 5.2: HospInfo content model ................................................................. 57
Figure 5.3: HospInfo basic rights ................................................................. 58
Figure 5.4: HospInfo navigational states (excerpt) ............................................ 58
List of Tables

Table 1.1: Tools integrated in the SDE .............................................................. 16

Table 3.1: FACPL Syntax ................................................................. 26
1 Introduction

One of the objectives of the NESSoS project is the use of a tool workbench enabling the integration of tools that support the development and analysis of secure software and systems. A variety of tools has been already developed or adapted by the partners of the project; others are still under construction. These tools are not only developed at different sites, but are also vastly different with regard to user interface, functionalities, required computing power, execution platform and programming language. However, all of them contribute to the development life cycle of secure software and might provide results which may serve as input to other tools. Thus, developers should be able to use some of them in combination generating so-called tool chains. The goal is to automate as many tasks as possible.

In NESSoS we follow a service-oriented tool integration approach, which enables developers to loosely integrate their tools as services to a workbench named Service Development Environment (SDE) – also called integrated platform. The SDE was developed within the SENSORIA project [54], a FET initiative funded by the EU from 2005 to 2010. It is currently maintained and extended by LMU within the scope of the NESSoS and the ASCENS [9] projects.

Although in a first instance of the SDE, only tools used in the project have been integrated, the aim of work package 2 (WP2) is also to consider security relevant tools that were developed outside the NESSoS project. The SDE platform is already publicly available online and open for tool contributions. All the so far integrated tool-wrappers are also available for download from the SDE website.

The deliverable D2.2 of the first reporting period has described the characteristics of the SDE core platform and has presented a set of tools integrated in the first period (see [25]).

The aim of this deliverable is to describe the challenges faced in integrating the engineering tools into the SDE, to provide an overview of available tools, a description of their application area and some use case examples. This material should enable SDE users to discover and seamlessly use all kind of tools integrated in the platform. The tool chain example makes clear that transformation services are required for converting tool in and output within the SDE.

SDE main characteristics. The SDE is based on Eclipse – an open source platform – and is open source itself. The SDE is an Integrated Development Environment (IDE) based on a Service-Oriented Architecture (SOA) approach, where each tool is represented as a service. Technically, the service-oriented OSGi framework is used. OSGi is based on so-called bundles, which are components grouping a set of Java classes and metadata providing among other things name, description and version. An OSGi bundle may provide arbitrary services to the platform and therefore all tools are integrated as bundles which offer certain functions for invocation by the SDE platform. Furthermore, it provides the ability to compose new tools out of existing ones, a process known as orchestration in the world of services.

The main features of the tool workbench are:

- The SDE Browser that contains a categorized listing of all tools which are currently available in this particular instance of SDE.
- The Function Browser that is available after double-clicking at one tool in the SDE Browser. It provides the description of the tool and its functions.
- The SDE Blackboard is used to store Java object values in-between service invocations when executing tool’s functions manually from within the SDE Browser.
- The SDE Shell is an orchestrator which can be used to employ JavaScript to call tool functions.
- The SDE Orchestrator provides graphical interface for creating tool chains by linking functions of (different) tools in order to create a new service that manages these tools, functions, inputs and outputs.

Tools to be used as part of the SDE must be implemented as OSGi bundles and contain a declarative description of the entry points of their functionality but are otherwise unlimited in their implementation.

---

1 SDE. http://www.nessos-project.eu/sde/
In particular,

- Tools may be written in Java and may consist of an arbitrary number of libraries, other Eclipse plugins, or external code.
- Tools may also wrap native code, thereby providing an interface to non-Java software.
- Tools may include functionality for calling remote services, thereby providing the link to Web services.

**Integrated tools.** In this reporting year, the tools AbsInt, PRRS, UWE2XACML, XACML2FACPL and X-CREATE were added to the SDE. Last year, Avantssar-atse (CL-ATSE), Avantssar Orchestrator, CORAS Tool, EOS (Eye OCL Software), Jalapa, MagicUWE, UML4PF and VeriFast were included. Table 1.1 provides an overview of all NESSoS tools integrated at the moment.

<table>
<thead>
<tr>
<th>Tool Name</th>
<th>Description</th>
<th>SDLC Phase</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>AbsInt</td>
<td>Static analysis of embedded systems</td>
<td>Testing</td>
<td>3.4</td>
</tr>
<tr>
<td>Avantssar-atse</td>
<td>Constraint logic based on attack searcher for security protocols and services</td>
<td>Testing</td>
<td>4.1 &amp; [25, 4.1]</td>
</tr>
<tr>
<td>Av. Orchestrator</td>
<td>Automatic orchestration of web services</td>
<td>service composition</td>
<td>4.2 &amp; [25, 4.2]</td>
</tr>
<tr>
<td>CORAS Tool</td>
<td>Model-driven approach to risk analysis</td>
<td>Risk/Cost Management</td>
<td>4.3 &amp; [25, 4.3]</td>
</tr>
<tr>
<td>EOS</td>
<td>Java component for efficiently performing the evaluation of queries written in OCL</td>
<td>Design</td>
<td>4.4 &amp; [25, 4.4]</td>
</tr>
<tr>
<td>Jalapa</td>
<td>Extension to the security model of Java that allows one to specify, analyze and enforce history-based usage policies</td>
<td>Implementation</td>
<td>4.5 &amp; [25, 4.5]</td>
</tr>
<tr>
<td>MagicUWE</td>
<td>CASE tool support for the development of web applications modeled with UWE</td>
<td>Design</td>
<td>4.7 &amp; [25, 4.6]</td>
</tr>
<tr>
<td>PRRS</td>
<td>Run-time management of security and dependability (S&amp;D) solutions and monitoring of the system context</td>
<td>Implementation</td>
<td>3.5 &amp; 4.6</td>
</tr>
<tr>
<td>UML4PF</td>
<td>Requirements analysis according to an enhanced version of Jackson’s problem frame approach</td>
<td>Requirements</td>
<td>4.8 &amp; [25, 4.7]</td>
</tr>
<tr>
<td>UWE2XACML</td>
<td>Conversion of UWE access control policies to policies written in XACML</td>
<td>Design &amp; Implementation</td>
<td>3.1</td>
</tr>
<tr>
<td>VeriFast</td>
<td>Verifier for single-threaded and multithreaded C and Java programs annotated with preconditions and postconditions written in separation logic</td>
<td>Implementation</td>
<td>4.9 &amp; [25, 4.8]</td>
</tr>
<tr>
<td>XACML2FACPL</td>
<td>Transformation of XACML policies to FACPL format</td>
<td>Design &amp; Implementation</td>
<td>3.3</td>
</tr>
<tr>
<td>X-CREATE</td>
<td>Automatic generation of test XACML requests from a XACML policy</td>
<td>Testing</td>
<td>3.2 &amp; 4.10</td>
</tr>
</tbody>
</table>

**Table 1.1: Tools integrated in the SDE**

Our first tool chain of MagicUWE, UWE2XACML and XACML2FACPL can be seen as a further tool, because users of the tool chain do not have to care which tools are executed in the background.

**Background/Foreground.** As written above, the SDE was developed within the SENSORIA project [54]. During NESSoS it was enhanced, as described in this deliverable and the integration of NESSoS tools and the tool chain are also foreground as they have not existed before.

**Relations to other work packages.** The CBK Search in the SDE (described in section 2.2) queries information from the CBK (WP5). Additionally, the integrated tools are used in the research work packages (WP6-WP10) and for the case studies (WP11).

**Structure of the deliverable.** Section 2 describes the SDE extensions recently implemented, which includes updates and management of examples. Section 3 presents tools integrated in the SDE in the current period of the project. Section 4 provides examples using already integrated tools in SDE and section 5 outlines possibilities for tool collaborations. Finally, section 6 summarizes and sketches future steps in the development of the NESSoS tool workbench and section 7 lists the NESSoS publications that are relevant for this work.
2 Extensions of the SDE

By the end of the first year of the NESSoS project, the SDE supported a uniform way to use tools and it already provided facilities for the execution of tool chains, which is the combined use of tools. At that time, full tool documentation was only available in the CBK [27], but not directly accessible from the SDE. Therefore, during the second year of the NESSoS project a CBK Search tool was implemented. It allows accessing CBK’s information directly from within the SDE, as described in section 2.2. Furthermore, getting familiar with unknown tools is made easier by additional assistance: instead of pure information, a mechanism was introduced to share and load examples (section 2.3).

2.1 Bugfixes and Updates

While maintaining a project, several bugs used to show up and used libraries might become outdated.

![Improved Error Handling](image)

Figure 2.1: Improved Error Handling

The GUI of the SDE Orchestrator can be used to execute tools in a row. However, error messages were shown, if a user, e.g., tried to execute a composed service, consisting of several method calls, without adjusting the properties first. Figure 2.1 shows the improved version. Additionally, hints are now given, as e.g. that the function names are not allowed to start with an upper case letter (“uweToFacplTransformation” is allowed, but not “Uwe2facplTransformation”, as can be seen in Figure 2.1). Several other issues in the SDE were fixed to improve the user experience.

Besides, the underlying Eclipse\(^1\) version was updated from Indigo to Juno, which made some adoption necessary. However, Indigo is used as common basis for NESSoS, but until now we have not encountered problems with Juno, except of difficulties, because of increased memory consumption. A bigger change came with the update of GMF (Graphical Modeling Framework\(^2\)), because the whole SDE Orchestrator had to be rebuilt, which means that the code is fully re-generated, except changed parts that have to be marked by a “@generated NOT” tag before. Somehow the generation of package names caused problems regarding spelling, but after some changes it worked well.

\(^1\)Eclipse. [http://www.eclipse.org/](http://www.eclipse.org/)

\(^2\)Graphical Modeling Framework. [http://eclipse.org/gmf-tooling/]
2.2 CBK Search

As the CBK is the main knowledge base of NESSoS, switching to the browser and logging in first was considered tedious. Therefore, we implemented a search for the CBK, as depicted in Figure 2.2. It can be installed in Eclipse separately and when a tool is opened in the SDE, a click on the NESSoS-CBK button opens the information of this tool in the CBK tab. Browsing the CBK manually is also possible in the CBK tab. For navigating forward or backward in the browser history, the green arrows next to the CBK logo can be used.

![CBK interface within the SDE](image)

**Figure 2.2: CBK interface within the SDE**

2.3 Managing Examples for Integrated Tools

The start using an unknown tool is always a difficult task. Besides good online tool documentation, the best way is learning by example. Following this principle, we have extended the SDE so that it supports example sets. The requirements for example sets are as follows:

- Execution of examples for tools and tool chains should be possible.
- Concrete input parameters should be automatically used.
- Examples should be easy to access.

Easy access to examples means that it should be possible to download example projects as zip-files from the SDE web page\(^3\) and to invoke a tool function using predefined input.

When executing a function of an integrated tool, a dialog shows up, which is depicted in Figure 2.3. There, the user can load a predefined SDE example set, by clicking on the button at the lower right.

---

\(^3\)SDE. NESSoS Software Workbench. [http://nessos-project.eu/sde](http://nessos-project.eu/sde)
Automatically, the new SDE extension searches the root of local Eclipse work spaces and a list with all possible parameters for the selected method is shown (Figure 2.4).

![Figure 2.3: Executing a method of a tool manually](image)

![Figure 2.4: Loading a SDE example set](image)

Technically, the tool owner has to create a new Eclipse project which is made available at the Web. Example files (inputs for the tools’ methods) and a ReadMe-file with further information about the usage of the tool have to be stored in this project. Finally, a description file with the extension .sdeExampleSet is added, which describes possible concrete parameters using the JSON format. For our example shown in Figure 2.4, a function of the integrated tool MagicUWE with the signature `editProjectWithMagicUWE(File project)` should be executed. Therefore, we have created an example set file with the following content:

```json
{
  "tool": "MagicUWE (in MagicDraw)",
  "version": "1.0.1",
  "functions": {
    "editProjectWithMagicUWE": [
      ["HospInfo.mdzip"],
      ["AddressBook.mdzip"]
    ]
  }
}
```
Several tool and function calls can be stored in one file. When files are referenced, as in this case, the path has to be given in relation to the location of the example set file. As parameters, types like Strings, Numbers, Booleans and Files could be used as well as Lists of them. Lists can also be nested. When using lists, the first value within a java.util.List-parameter has to start with the full type of the list elements, followed by "", as e.g. java.util.List[Ljava.lang.String; Hello, Bye, See you]. Nested lists just need this type to be specified for the inner elements.
3 Further Integrated Tools

The tools integrated in SDE during the first year of the NESSoS project were presented in Deliverable D2.2 [25]. These tools are Avantssar-atse (CL-ATSE), Avantssar Orchestrator, CORAS Tool, EOS (Eye OCL Software), Jalapa, MagicUWE, UML4PF and VeriFast. In this section, we describe a set of further security relevant tools, which were integrated during the second year of the project. These tools are UWE2XACML, X-CREATE, XACML2FACPL, PRRS and AbsInt. X-CREATE is a tool for the automatic generation of test inputs from a XACML policy. UWE2XACML is a tool for transforming access control models represented with the UML-based Web Engineering (UWE) modeling language into XACML policies. XACML2FACPL supports transformation of XACML policies into policies written in Formal Access Control Policy Language (FACPL) with the purpose of evaluation and verification. The AbsInt tool suite for the static analysis of embedded systems. Finally, the Platform for Runtime Re-configurability of Security (PRRS) is a tool that provides run-time management of Security and Dependability (S&D) solutions and monitoring of the system context.

Figure 3.1 shows the mapping of all integrated tools to the Software Development Lifecycle (SDLC). This schema is an extended version of the one presented in Deliverable D2.1 [24].

Figure 3.1: Overview of integrated tools

This overview of the available tools shows the phases of the SDLC in which the tools are applicable. It has the advantage to enable the identification of possible tool chains. These tool chains usually make use of tools that are either located in the same phase or in subsequent phases. Tools supporting model transformations are usually located between phases, e.g. the tool UWE2XACML that converts UWE to XACML, i.e. from a graphical modeling language for designing web applications to access control policies expressed in the standard XACML.

An example for a tool chain using a similar phase is the tool chain that combines UWE2XACML and XACML2FACPL, which means that certain models are translated into other formats. Tool chains with tools from different phases are also conceivable, e.g., we plan to connect UWE2XACML and X-Create to be able to generate tests from XACML policies. More information about tool chains and a detailed example can be found in chapter 5.
3.1 UWE2XACML

UWE2XACML [26] is a tool for transforming access control modeled using the UML-based Web Engineering (UWE) language into XACML policies. UML-based Web Engineering (UWE) is a graphical, modeling language. eXtensible Access Control Markup Language (XACML) is the OASIS standard eXtensible Access Control Markup Language used for the definition and enforcement of policies. This kind of transformation tool provides mechanisms that enables developers to use a high-level language to specify access control policies and transform them to a standard language instead to use an Extensible Markup Language (XML) language, which is time-consuming to write and difficult to read.

3.1.1 Features

The algorithm transforms not only the UWE basic rights model, but also states with a roles tag from the navigation states model. The aim is to constrain whether or not a user is allowed to navigate to a certain area. The UWE basic rights model is used to express the role based access control on the domain concepts specified in the content model, picking the roles from a user model. The navigation state model provides a graphical representation of the path the user can navigate in the system. This model also represents security features as, e.g., authentication, access control and secure connections. The content model represents the domain concepts that are relevant for the web application and the relationships between them. The role model defines a hierarchy of user groups with the purpose of authorization and access control. It is usually included in a user model, which specifies basic structures as e.g. that a user can take on certain roles simultaneously.

Intuitively, the transformation generates a XACML PolicySet for each role, each of which contains one Policy for any class connected to the considered role. Cf. excerpt of transformation result in Figure 3.2 provided via the SDE. Furthermore, a single Policy is used to deny access to all resources not specified in the PolicySet, which is the default behavior of UWE’s basic rights models.

![Figure 3.2: UWE2XACML result in SDE](image-url)

Notably, to allow a sub-role of a given role to use the permission specified by the super-role, the target of the PolicySet corresponding to the super-role is extended to also match requests from the sub-role. Each Policy for a constrained class contains one Rule for each action between the role and the class. Attributes targeted by *All actions are divided into a set of Resources, omitting those from the {except} tag. Object Constraint Language (OCL) constraints inside UML comments with authorizationConstraint
stereotype are transformed to a Condition. The condition is located within a Rule representing the appropriate action. For the time being, we have implemented only a few basic OCL constraints.

3.1.2 Integration into the SDE

Technically, the UWE2XACML transformation from projects modeled with the UWE Profile to XACML 2.0 is implemented using the modeling framework of Eclipse which facilitated the integration in the SDE. We also used Xpand 1.2.1\(^1\), a language specialized on code generation based on models defined by the modeling component of Eclipse. Xpand is based on workflows, which apply templates in order to parse the model and to produce the desired code.

To be able to use the project files of MagicDraw 16.8 for the transformation with Xpand, they have to be exported as Eclipse UML2 (v3.x) XML Metadata Interchange (XMI) file. For complex tasks as the transformation to XACML, Java extensions are used from within the Xpand templates, because Java enables us, e.g., to group several dependencies with equal constraints to only one Rule.

The technical details of the integration in the SDE are briefly described in the following:

<table>
<thead>
<tr>
<th>Technical Requirements</th>
<th>XPand has to be installed</th>
</tr>
</thead>
<tbody>
<tr>
<td>License</td>
<td>Eclipse Public License - v 1.0</td>
</tr>
<tr>
<td>Eclipse Update Site</td>
<td><a href="http://svn.pst.ifi.lmu.de/update/NESSoStools/UWE2XACML/">http://svn.pst.ifi.lmu.de/update/NESSoStools/UWE2XACML/</a></td>
</tr>
<tr>
<td>Installation Guide</td>
<td>Before using the Eclipse's Update Manager to install the wrapper, XPand has to be installed, preferably via &quot;Install Modeling Components&quot;.</td>
</tr>
<tr>
<td>Tool input type</td>
<td>fileName : String – reference to UWE project in XMI format</td>
</tr>
<tr>
<td>Tool output type</td>
<td>xacmlFile : File – a XACML 2.0 file containing the policies modeled with UWE</td>
</tr>
</tbody>
</table>

Within the UWE2XACML wrapper the method file transformUwe2xacml(String fileName) throws IOException transforms the UWE model to a XACML policy file.

3.2 X-CREATE

X-CREATE [21, 20, 19] is a tool for the automatic generation of test inputs from a XACML policy. The tool X-CREATE (XaCml REquests derivAtion for TEsting) implements different strategies for deriving XACML requests from a XACML policy. These strategies are based on combinatorial analysis of the values specified in the XACML policy. X-CREATE also derives a test suite, covering the XACML Context Schema that describes the overall structure of the XACML input requests. The aim of the derived XACML requests is twofold: testing of policy evaluation engines and testing of access control policies.

3.2.1 Features

X-CREATE targets the automatic generation of XACML requests by providing the user with different testing strategies. The high-level architecture of X-CREATE is organized on two main components: the Policy Analyzer and the Request Generator. In addition, a GUI component helps users in the testing strategies application and a database stores all required data for the requests derivation.

The X-CREATE GUI lets the user select a XACML policy. The Policy Analyzer component takes as input this XACML policy and by a XQuery parser derives four values sets (called SubjectSet, ResourceSet, ActionSet and EnvironmentSet) containing the subjects, resources, actions and environments policy values respectively. The parser results are stored in a XML file which preserves the tree hierarchy of the policy nodes (policy targets, rule targets and conditions). For negative and robustness testing, the Policy Analyzer adds random values to the SubjectSet, ResourceSet, ActionSet, and EnvironmentSet sets.

The values of the SubjectSet, ResourceSet, ActionSet, and EnvironmentSet are combined using a combinatorial approach.

Once the combinations of policy values are generated, by means of the GUI, the user can choose the testing strategy to be applied (see Figure 3.3) among:

\(^1\)Xpand. http://wiki.eclipse.org/Xpand
• **Simple Combinatorial** testing strategy [19]: for each obtained combination a simple request containing the values of that combination is generated.

• **XPT-based** testing strategy: the obtained combinations are used for filling according to the request values assignment described in [21], the requests structures obtained applying the XPT-strategy to the XACML context schema.

• **Hierarchical Simple Combinatorial**: the main idea of this strategy is to reduce the number of test cases when testing is focused on a single policy or rule. This strategy consists on deriving the SubjectSet, ResourceSet, ActionSet, and EnvironmentSet by considering only the values that belong to a subtree deep-rooted in one of the policy nodes selected by the X-CREATE GUI. As for **Simple Combinatorial** strategy, for each obtained combination a simple request containing the values of that combination is generated.

• **Hierarchical XPT-based**: as in **Hierarchical Simple Combinatorial**, the SubjectSet, ResourceSet, ActionSet, and EnvironmentSet are generated starting by a subtree deep-rooted and the values combinations are assigned to the requests structures as described in **XPT-based** testing strategy.

X-CREATE automatically generates a set of XACML requests accordingly with the test strategy selected. By the X-CREATE GUI the user chooses the number of derived requests. In case of **Simple Combinatorial** and **Hierarchical Simple Combinatorial** the user inputs the number of desired requests as showed in Figure 3.3. If **XPT-based** or **Hierarchical XPT-based** testing strategies are selected, the user has the possibility to choose the requests structures to be filled with the policy values.

The online version of the tool can be accessed from the X-CREATE Home Page².

---

²X-CREATE Home Page. [http://labse.isti.cnr.it/tools/xcreate](http://labse.isti.cnr.it/tools/xcreate)
3.2.2 Integration into the SDE

The technical details of the integration in the SDE are described in the following:

<table>
<thead>
<tr>
<th>Technical Requirements</th>
<th>License</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java 1.6 or higher and MySQL DBMS 5.1.</td>
<td>GNU Lesser General Public License (L-GPL). See more information about this license at <a href="http://www.gnu.org/copyleft/lesser.html">http://www.gnu.org/copyleft/lesser.html</a>.</td>
</tr>
</tbody>
</table>

**Eclipse Update Site**


**Installation Guide**

The preferred approach to install the plugin is by using the Eclipse's Update Manager.

**Tool input type**

- XACML 2.0 policy file (*.xml).

**Tool output type**

- ArrayList of type java.lang.String containing the derived set of XACML requests.

The current X-CREATE tool integrated into the SDE implements the *Simple Combinatorial* testing strategy described above. Figure 3.4 shows the main function and options of the X-CREATE tool integrated in the SDE view.

![Figure 3.4: X-CREATE as a SDE component](image)

**3.3 XACML2FACPL**

XACML2FACPL [26] is a tool for transforming XACML policies into policies written in FACPL, which is a manageable alternative syntax to XACML through a BNF-like grammar.
3.3.1 Features

The FACPL syntax is reported in the following. As usual, square brackets are used to indicate optional items.

\[
Policies ::= \{ \text{Alg}; \text{target} : \{ [ \text{Targets} ] \}; \text{Policies} \} \\
| \{ \text{Alg}; \text{target} : \{ [ \text{Targets} ] \}; \text{rules} : \{ \text{Rules} \} \} \\
| \text{Policies} \text{ Policies}
\]

\[
\text{Alg ::= deny-overrides} \mid \text{permit-overrides} \mid \ldots
\]

\[
\text{Targets ::= MatchId(value,name)} \mid \text{Targets } \lor \text{Targets} \\
| \text{Targets } \land \text{Targets} \mid \text{Targets } \cap \text{Targets}
\]

\[
\text{MatchId ::= string-equal} \mid \text{integer-equal} \mid \ldots
\]

\[
\text{Rules ::= (Effect [; target : \{ \text{Targets} \}][; condition : \{ expr \} ] )} \\
| \text{Rules } \text{ Rules}
\]

\[
\text{Effect ::= permit} \mid \text{deny}
\]

Table 3.1: FACPL Syntax

FACPL policies can be simple policies of the form \( \{ \text{Alg}; \text{target} : \{ [ \text{Targets} ] \}; \text{rules} : \{ \text{Rules} \} \} \) or, recursively, policy sets of the form \( \{ \text{Alg}; \text{target} : \{ [ \text{Targets} ] \}; \text{Policies} \} \). Both policies and policy sets specify the algorithm for combining the results of the evaluation of the contained elements to which the policy/policy set applies. The first part of a result from an example executed within the SDE is depicted in Figure 3.5.

A \text{target} identifies the set of access requests that a rule, a policy or a policy set is intended to evaluate. Specifically, a target specifies the set of subjects, resources, actions and environments to which the corresponding rule/policy/policy set applies. In the XML-based syntax of XACML, the target element may contain four separate elements, one for each of the above categories.

To obtain a more compact notation, FACPL represents a target as an expression built from match elements, i.e. terms of the form \( \text{MatchId(value,name)} \), by exploiting an operator for logical disjunction, \( \lor \), and two operators for logical conjunction, \( \land \) and \( \cap \). Each match element spells out a specific value that the subject/resource/action/environment in the decision request (identified by a name) must match, according to a given matching function.
Anyway, this target representation does not lead to a loss of information, because names can be structured and hence, as shown before in the designator example, can include the corresponding category. In a match element, **MatchId** specifies the (boolean) matching function to be used to compare the given literal value with the value of the attribute identified by the given **name**. If a target is empty, the corresponding rule/policy/policy set applies to any request.

### 3.3.2 Integration into the SDE

The transformation, performed by the XACML2FACPL tool is straightforward. Its flow loops over the policy sets creating the necessary data structures for the FACPL representation. The original XML document is read by using JAXB\(^3\). The loop over the elements is driven by the XACML schema definitions by traversing its data types.

The technical details of the integration in the SDE are briefly described in the following:

<table>
<thead>
<tr>
<th>Technical Requirements</th>
<th>No special requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>License</strong></td>
<td>Eclipse Public License - v 1.0</td>
</tr>
<tr>
<td><strong>Eclipse Update Site</strong></td>
<td><a href="http://svn.pst.ifi.lmu.de/update/NESSoStools/XACML2FACPL/">http://svn.pst.ifi.lmu.de/update/NESSoStools/XACML2FACPL/</a></td>
</tr>
<tr>
<td><strong>Installation Guide</strong></td>
<td>The preferred approach to install the plugin is by using the Eclipse's Update Manager.</td>
</tr>
<tr>
<td><strong>Tool input type</strong></td>
<td>xacmlFile : File – XACML policy file</td>
</tr>
<tr>
<td><strong>Tool output type</strong></td>
<td>facplFile : File – FACPL policy file</td>
</tr>
</tbody>
</table>

Within the XACML2FACPL wrapper the method `File transformXacml2facpl(File xacmlFile) throws Exception` transforms a XACML policy file to a policy file in the FACPL format.

### 3.4 AbsInt

The AbsInt\(^a^3\)\(^4\) is a suite of industrial-strength tools for the static analysis of embedded systems. In particular, \(a^3\) comprises tools (called aiT and TimingExplorer) for the estimation of worst-case execution times based on the static cache analysis by Alt et al. [7]. The tools cover a wide range of processors such as ERC32, TriCore, M68020. LEON3 and several PowerPC models (aiT), as well as models with freely configurable caches (TimingExplorer). Although AbsInt is not a security related tool, but a tool to evaluate the performance of embedded software, it has been shown that it can be used to quantify the leakage of side-channels that arise due to CPU caches (see [44]).

In fact, many modern computer architectures use caches to bridge the latency gap between the CPU and main memory. Caches are small, fast memory that store the contents of previously accessed main memory locations; they can improve the overall performance because typical memory access patterns exhibit locality of reference. On today’s architectures, an access to the main memory (i.e. a cache miss) may imply an overhead of hundreds of CPU cycles w.r.t. an access to the cache (cache hit).

While the use of caches is beneficial for performance, it can have negative effects on security: An observer who can measure the time of memory lookups can see whether a lookup is a cache hit or miss, thereby learning partial information about the state of the cache. This partial information has been used for extracting cryptographic keys from implementations of AES [18, 51, 36], RSA [52], and DSA [6]. In particular AES is vulnerable to such cache-attacks, because most high-speed software implementations make heavy use of look-up tables. Cache attacks are the most effective known attacks against AES and allow recovering keys within minutes [36].

In [44] a method for establishing formal security guarantees against cache-attacks is proposed. The guarantees obtained are upper bounds on the amount of information about the input that an adversary can extract by observing the CPU's cache state after execution of the program; they are based on the actual program binary and a concrete processor model and can be derived entirely automatically. At the heart of that approach is a novel technique for effective model-counting that enables one to connect state-of-the-art techniques for static cache analysis and quantitative information-flow analysis. More precisely, a concise implementation of counting procedures in Haskell [5] is given and this counting engine is connected to

\(^3\)JAXB. http://jaxb.java.net

the AbsInt $a^3$ [1] output. Using this tool chain, an analysis of a binary implementations can be done with respect to the leakage of both data and instructions caches. For example, a 128-bit AES from the PolarSSL library [4] was analyzed with and without countermeasures to leakage in place, based on a 32-bit ARM processor with a 4-way set associative data cache with LRU replacement strategy and different cache sizes.

### 3.4.1 Features

The TimingExplorer receives as input a binary program and a cache configuration and delivers as output a control flow graph in which each (assembly-level) instruction is annotated by the corresponding abstract `may` and `must` information, where memory locations are represented by strings, abstract cache lines are lists of memory locations, abstract sets are lists of abstract lines, and abstract caches are lists of abstract sets. The AbsInt SDE plugin allows one to extract the annotations of the final state of the program, and provide them as input to the counting engine devised in [44] that allows to computes upper bounds to the leakage of the programs inputs to diverse adversaries exploiting the CPU cache as a side-channel.

![Figure 3.6: Overview of the AbsInt SDE Plugin](image)

The AbsInt SDE plugin receives the location of an AbsInt `.apx` project in input and can be configured to set the location of the AbsInt tool itself and the directory where the results will be stored. It is usually located within the user’s home directory. In Figure 3.6 a view of the plugin with default configurations values is depicted. The `.apx` project file contains the path where the binary and the cache configuration files are located, as well as the information relative to the last executable instruction of the binary. The SDE plugin will then call the AbsInt tool in batch mode with this configuration information and will read the output of the tool, containing the abstract cache states for the last executable instructions for further processing. Figure 3.7 contains an example of such an abstract state for the must analysis.

For instance, in [44] the AES C source code is compiled into a binary for the ARM7TDMI [2] processor using the GNU ARM GCC compiler [3]. Although the original ARM7TDMI does not have any caches, the AbsInt TimingExplorer supports this processor with the possibility of specifying arbitrary configurations of data/instruction/mixed caches with a particular replacement strategy (LRU). For their experiments they use data caches with sizes of 16-128 KB, associativity of 4 ways, and a line size of 32 Bytes, which are common configurations in practice. They analyze this implementation with and without the preloading countermeasure applied and for two different adversary models, obtaining the following results.
Without preloading, the derived upper bounds for the leakage in one run (about the payload and the key) exceed the size of the key and are hence too large for practical use. With preloading and a powerful adversary model, however, the derived bounds drop to values ranging from 55 to 1 bits, for cache sizes ranging from 16KB to 128KB. For a less powerful but realistic adversary the bounds drop even further to ranges from 6 to 0 bits, yielding strong security guarantees. This case study shows that the automated, formal security analysis of realistic cryptosystems and accurate real processor models is in fact feasible.

### 3.4.2 Integration into the SDE

The technical details of the integration in the SDE are briefly described in the following:

<table>
<thead>
<tr>
<th>Technical Requirements</th>
<th>AbsInt has to be installed separately.</th>
</tr>
</thead>
<tbody>
<tr>
<td>License</td>
<td>Eclipse Public License - v 1.0 (the wrapper), proprietary (the tool AbsInt itself)</td>
</tr>
<tr>
<td>Eclipse Update Site</td>
<td><a href="http://svn.pst.ifi.lmu.de/update/NESSoSTools/AbsInt/">http://svn.pst.ifi.lmu.de/update/NESSoSTools/AbsInt/</a></td>
</tr>
<tr>
<td>Installation Guide</td>
<td>The preferred approach to install the plugin is by using the Eclipse's Update Manager.</td>
</tr>
<tr>
<td>Tool input type</td>
<td>absIntProjectFilePath : String</td>
</tr>
<tr>
<td>Tool output type</td>
<td>absIntResult : File</td>
</tr>
</tbody>
</table>

Within the AbsInt wrapper a method called “executeAbsInt” is available with the following signature: `File executeAbsInt(String projectFile) throws AbsIntException`. The option `ABSINT_EXECUTABLE_LOCATION` points to the installed AbsInt executable and `ABSINT_OUTPUT_LOCATION` references a folder where the output is stored.
3.5 PRRS

The Platform for Runtime Re-configurability of Security (PRRS) [35, chap. 2.3] is a tool that provides run-time management of Security and Dependability (S&D) solutions and monitoring of the system context.

![Figure 3.8: PRRS use cases](image)

3.5.1 Features

PRRS allows maintaining the safety and reliability of an application (PRRS-aware application), providing security solutions from a repository (S&D Library) and monitoring these solutions according to present rules (see Figure 3.8).

PRRS uses two databases called ‘sdlibrary’ to store all the S&D solutions and patterns and ‘contextmanager’ to store context information. The scripts to create the required tables are located in the folder databases. When executing `mysq1 -u root` in this folder we can use `source sdlibrary.sql` and `source contextmanager.sql`. Furthermore, it is also need to create a specific user ‘prrs’ that grants all privileges to access those databases:

```
mysql> create user 'prrs'@'localhost' identified by 'prrs';
mysql> grant all privileges on `sdlibrary`.* to 'prrs'@'localhost' with grant option;
mysql> grant all privileges on `contextmanager`.* to 'prrs'@'localhost' with grant option;
```

The PRRS_installation\conf folder contains the SandDLibrary.conf and ContextDB.conf files, where the configuration of recently created databases can be modified, as e.g., URL, user name and password.

The ExecComps folder is the folder where we store the available S&D Solutions, in our case there is only one solution: EncryptAES, which is shown in Figure 3.9.

The server is started using the karaf.bat script from the bin folder. It starts all the bundles located within the deploy folder.

When server is started it launches a console to be able to add the patterns and solutions in the database. This console interface will open in another window. It can be used to update the sdlibrary database with new patterns and S&D Solutions using Tools->Add S&D Artifact, as shown in Figure 3.10.

---

5The code has been tested with mysql5.5.8 installed with WampServer. [http://www.wampserver.com/en/](http://www.wampserver.com/en/)
To use the server from the SDE client (as shown in section 4.6), we include the following files located in the folder `PRRS\installation\execComps\EncryptAES\xml`: EncryptClass.xml, EncryptPattern.xml and EncryptAESImplementation.xml. It is important that they are added in the following order: class, pattern and implementation. Exemplarily, the inclusion of the class file is shown in Figure 3.11.
3.5.2 Integration into the SDE

The technical details of the integration in the SDE are briefly described in the following:

Technical Requirements
Java 1.6 and MySQL 5.5.8. A java.policy file with grant AllPrivileges is needed in the Eclipse home directory.

License
Free SDE wrapper, proprietary PRRS tool

Eclipse Update Site
http://svn.pst.ifi.lmu.de/update/NESSoTools/PRRS/

Installation Guide
PRRS server (PRRS_server_karaf_X.X.zip) previously running. Unzip the file and follow the instructions to create the databases and start the server. The tool is an example client that can be installed using the Eclipse Update Manager.

Tool input type
2 String parameters: text and keyword.

Tool output type
Boolean. Other results are displayed in server’s console

Within the PRRS wrapper a methods with the signature: boolean encrypt(String textPlain, String password) is available. Its usage is explained in further detail in section 4.6. Basically this SDE tool is an example of a PRRS-aware application. Those applications are PRRS clients that request a solution provided by PRRS server. The PRRS-aware applications can be developed using the application development support libraries included in the source of this tool.
4 Usage Examples of Integrated Tools

As introduced in section 2.3, an extension of SDE was implemented to support example sets in the second year of the NESSoS project. For some NESSoS tools example sets are already provided which can be downloaded from the SDE website.1

In general, examples that come with read-me files or tutorials provide additional help for the use of new tools without having to read a full specification first. Furthermore, the SDE facilitates the use of new tools, as each tool can be installed using an Eclipse update site. In case the tools have further requirements, e.g. regarding additional software that has to be installed first, users can find installation instructions in the CBK. In addition, executing functionality in the SDE has the advantage to be uniform, as each tool appears in the tool browser of the SDE. Double-clicking on a tool will cause the display of the available functions in the function browser. There, each function can be selected and executed with given parameters or with parameters from an example set. More detailed information about working with the SDE can be found in the SDE tutorial [53].

4.1 Avantssar-atse (CL-ATSE)

CL-AtSe [55, 8, 11] is a Constraint Logic based Attack Searcher for security protocols and services. The main idea in CL-AtSe consists in running the protocol or set of services in all possible ways by representing families of traces with positive or negative constraints on the intruder knowledge, on variable values, on sets, etc. Thus, each run of a service step consists in adding new constraints on the current intruder and environment state, reducing these constraints down to a normalized form for which satisfiability is easily decidable, and decide whether some security property has been violated up to this point. CL-AtSe does not limit the service in any way except for bounding the maximal number of times a service can be iterated, in the case such an iteration is allowed in the specification. Otherwise, the analysis might be non-terminating on secure services and only heuristics, approximations, or restrictions on the input language could lift this limitation.

4.1.1 Features

Input CL-AtSe reads any specification written in ASLan by default. ASLan is a formal language for specifying trust and security properties of services, their associated policies, and their composition into service architectures. A service's behavior (or protocol role) in ASLan is defined by three elements: a set of transitions that can change facts from the current state to another, including messages to be exchanged (with applied security policies) as first order terms; some initial state — a set of facts; and a finite set of Horn clauses typically used to define an authorization logic.

The security goals to verify can be expressed in two ways: as an attack state and Linear Temporal Logic (LTL) goal. The former is represented as a pattern of a state that is considered insecure (if the tool is able to attain such global state that conforms with this pattern, an attack is reported). The latter is more general and expressive: if the tool is able to satisfy the given LTL formula it will report an attack. However, the tool has currently a limited support of LTL goals.

Output If a security property of the input specification is violated then CL-AtSe outputs a warning (UNSAFE), details about the analysis (e.g. whether the considered model is a typed or an untyped one), the property that was violated (secrecy, for instance), statistics on the number of explored states, and, finally, an ATTACK TRACE that gives a detailed account of the attack scenario. If no attack was found then similar information is provided (except UNSAFE and the ATTACK TRACE).

Simplifications and optimizations CL-AtSe implements modules to simplify and optimize the input specification statically before analysis. The simplification module aims at reducing the search space without truly changing the structure of the problem to be analyzed. More aggressively, the optimization module tries to change the structure of the input specification to pre-process a significant part of the

---

1SDE website. http://nessos-project.eu/sde
branching that would otherwise be done during analysis, so that many dead-ends can be found and removed statically.

**Suspending the analysis**  CL-AtSe is capable to stop and restart later an analysis, by writing and reading a state file containing the current state of the analysis. This can be produced either anytime during a very long analysis to resume, e.g., in case of power loss; this can also be used each time an attack is found to store the state of the system without the attack, and thus, resume the analysis to find other attacks (either on the same branch with other constraints, or on other branches). The latter is used by the AVANTSSAR Orchestrator (See [13]) to enumerate orchestrations. CL-AtSe takes care not to produce new attacks which would admit an attack already found as a prefix. Finally, this is also used for massive parallel computing, by first producing one state file for each sub-state of the model, and then analyzing them all in parallel.

**Non-Deductibility constraints**  Recently, CL-Atse has been extended with non-deductibility constraints. This means that now, the user can define Aslan v1 constraints requiring that the intruder runs a transition at the condition that he cannot deduce something (instead of the usual positive-only constraints on his knowledge). This difficult problem comes primarily from specific orchestrations models, where the mediator must be generated with the guaranty that some data belong unknown, and even more, non-deductible, to him. This can also be used to help modeling separation of duty.

### 4.1.2 Working with CL-Atse in SDE

CL-Atse can be queried in many different ways. First, naturally you can download the tool from the CASSIS web pages\(^2\) directly. However, CL-Atse is also included in packages with other tools, like the Avantssar Platform. The advantage of such distribution is that you get a translator to produce Aslan v1 files from an high-level Aslan v2 specification. See the Avantssar homepage\(^3\) for documentations on Aslan v2 and the translation tools. CL-Atse also accepts old IF specification files as produces by the Hipsi2IF translator from the Avispa project\(^4\)). Moreover, you can also query CL-Atse online directly from our server, with no need for downloading and installing the tool, either directly at the CASSIS web page, or using the online interfaces provided by each of the projects cited above. For integrating CL-Atse with other tool online, you can also query it as a web service using the WSDL files we offer. Also, it has been integrated into NESSoS's SDE. Finally, an Aslan GUI has been developed in Nancy for producing Aslan v2 (and thus Aslan v1) files much more easily and with limited knowledge of the language. This is described below.

**The Aslan Graphical User Interface**  A tool called Aslan GUI has been developed in Nancy to help modelers in producing Aslan v2 (alias Aslan++) specifications files without the need for a deep and complete knowledge of this language. Global knowledge of the language is still necessary to produce meaningful specifications, but, the interface takes care of the details like e.g. the syntax. Therefore, the user just needs to fill boxes, add agents and roles, describe each protocol or Web Service step, etc.. (see e.g. Figure 4.1). The GUI shows the specification as a Message Sequence Chart to convince the user that he did it right. It also automatically exports the specification to Aslan v2, translate it to Aslan v1, query CL-Atse properly, re-imports the attack found (if any) through the translator, and shows it as another Message Sequence Chart to the user which can then see the vulnerability of his model. The GUI, written in Python primarily for Linux, can be downloaded from Cassis’s server.

### 4.2 Avantssar Orchestrator

The AVANTSSAR Orchestrator [13] is a tool for automatic orchestration of Web Services taking into account their security policies. In short, it generates a service called mediator that is able to satisfy requests of a given client with the help of given community of available services. Therefore, what we call “Orchestrator” is a service (or tool) that reads the ASLan models of the client plus other services, and produces

\(^2\)CASSIS team at LORIA \(\text{http://cassis.loria.fr/}\)
\(^3\)Avantssar home page. \(\text{http://www.avantssar.eu/}\)
\(^4\)Avispa project. \(\text{http://www.avispa-project.org/}\)
Figure 4.1: Aslan GUI's main page (defining the Environment)
a new service called “mediator”. The mediator can query services, deduce new knowledge, and create new messages to adapt the client and services interfaces. And we call “orchestration” the safe interaction between the “mediator” and the client plus other services that fulfill all goals and security constraints.

The orchestration approach relies on the analogy with a state reachability problem in cryptographic protocols analysis domain. Exploiting this idea, the Orchestrator is based on CL-AtSe, a tool for solving cryptographic protocols insecurity in a presence of an active Dolev-Yao intruder.

The online version of the tool can be accessed from the CASSIS website5.

4.2.1 Features

Input  The input problem is described in the ASLan language [12], extended with some keywords in order to distinguish agents like, e.g., the client, some service, etc. The client is a special agent, since the tool must automatically generate a mediator able to satisfy each client’s requests. The mediator reaches his goal by invoking and composing available services, and process himself parts of the messages when possible. Thus, the mediator is not just a place-holder for composing services, but also an active agent with Dolev-Yao capabilities. The available services and the client are specified in a form of transition systems in the ASLan language. The mediator is defined only by its initial knowledge. Another type of input is admitted: instead of specifying a client, one may partially define a mediator by providing only the part related to the communication with the putative client.

The tool also accepts session-id, an identifier of a job, to continue solving the previously defined problem. Besides the orchestration problem, a modeler, using ASLan syntax, can add some global security properties that can be validated after solving the orchestration problem. These properties are checked by creating automatically a new specification model from the orchestrator’s output. This model describes the client, the generated mediator, the services, and the security properties. Then, along all possible orchestration’s outputs, one is chosen for which the analysis with CL-AtSe of this model finds no attack.

Output  In the case where the specification of the client is given, an ASLan specification of the mediator that satisfies the client’s requests is produced. In the case where the partial specification of the mediator is given, a specification of the putative client is generated. Moreover, a new mediator service is issued, which extends the mediator given in the input with the necessary interactions with the available services.

The tool also outputs the session-id as required above to build other solutions of the same problem. The global security properties are preserved, thus, the output specification is ready to be analyzed for detecting vulnerabilities.

Generating different solutions  Since the generated mediator can be vulnerable with regard to specified global security properties in the presence of active Dolev-Yao intruder, it may be rejected by the modeler. Then, the user possibly wants to generate another mediator and check it again for the vulnerabilities using, e.g., CL-AtSe. This recurring process is automated by the tool, and relies on CL-Atse’s capability to resume after finding an attack and negating it’s conjunction of constraints.

Deep link with CL-Atse  Even if invisible for the end-user, it is interesting to know that the Avantssar’s orchestrator relies in fact completely on CL-Atse for solving constraints on traces. That is, the orchestrator transforms the orchestration problem (client-based or goal-based) into an analysis problem understandable by CL-Atse, where the mediator becomes the intruder, and where satisfying the client is an insecurity property. Then CL-Atse finds an attack, which is a trace for the intruder where the client is satisfied, i.e. the result of the orchestration once rewritten into a new role called mediator. The consequence of this for the user is that, CL-Atse and the Orchestrator shares necessarily the same model of protocols and web-services.

4.2.2 Working with the Orchestrator in SDE

The Avantssar’s Orchestrator can be queried in many different ways. First, like CL-Atse, it can naturally be downloaded6 or queried online like a Web service using the WSDL files, and it is also integrated in

5CASSIS team at LORIA http://cassis.loria.fr/
NESSoS’s SDE. However, the most common way to query the Orchestrator is through the Avantssar’s online interface, because this automates the process and analyzes the generated mediator as soon as it is produced.

Since the input language is Aslan (i.e. Aslan v1) like for all tools in the Avantssar’s Platform except translators, the Aslan GUI can help in designing an orchestration model as long as the user is aware of orchestration-specific elements that must be included (i.e. identify the client, give the mediator’s initial knowledge, ..).

Non-deductibility constraints Thanks to CL-Atse’s recent improvement for running transitions with non-deductibility constraints, the Avantssar’s Orchestrator can now directly generate orchestrations where the mediator is unable to deduce e.g. some message. This is a great help to produce valid orchestrations w.r.t. security policies, since invalid mediators can now be skipped directly by the orchestrator instead of being later rejected by the analysis tool. This shortens the process.

The Loan Origination Process sample This simple orchestration case-study is a sample of Orchestrator usage that also shows the interest of adding the support for non-deductibility constraints in CL-Atse. It is depicted in Figure 4.2. The idea is the following: a Client needs to obtain some loan from his bank. However, the bank wants the client’s request for a loan to be studied and accepted by two clerks before continuing. These clerks can be chosen in a pool of clerks, but, the client must not know which clerks really study his request. Therefore, it is up to some mediator to link the client with the clerks, and this mediator must guaranty various properties. In particular, it must choose clerks in a fair way and hide their identities to the client; it must guaranty to the client that his request is properly transmitted; And it must also prove that it cannot know the amount of the loan. This is this last point that requires us to use non-deductibility constraints. We also used such constraints for coding separation of duty between clerks. Given the length of the specification, we cannot reproduce it here directly. However, it is visible online at the CL-Atse wiki.

One specificity of the Loan Origination Process (LOP) model is that, technically, the mediator could assume himself the role(s) of a Clerk. According to the initial knowledge and the data transmitted, this would be possible. However, the security policies forbid it, and thus, it is up to the Orchestrator to generate a non-trivial mediator that avoids doing this (and also avoids to be in a situation where he could deduce the loan amount). To show the behavior of the orchestrator in this situation, a few variants of this model have been analyzed :

<table>
<thead>
<tr>
<th>Clerk</th>
<th>Clerk</th>
<th>Mediator</th>
<th>Client</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>Advertises to be a Clerk</td>
<td>Choices two agents as Clerks</td>
<td>Wants a loan of some Amount</td>
</tr>
<tr>
<td>...</td>
<td>Receives a request</td>
<td>Transmits queries and responses</td>
<td>Loan must be checked by Clerks</td>
</tr>
<tr>
<td>...</td>
<td>Answers that request.</td>
<td>Must ignore data like Amount.</td>
<td>Produces queries Expects answers</td>
</tr>
</tbody>
</table>

Figure 4.2: Improved Avantssar’s LOP case-study

LOP model without neg. constraints and without any Clerk In this situation, the Orchestrator finds a mediator which assumes himself the role of a Clerk (since no neg. constraints are here to prevent it). While an orchestration is found, further analysis w.r.t. the security policies will reject this mediator. This primarily shows that the mediator really has the knowledge to act as a (fake) Clerk.

LOP model with neg. constraints but without any Clerk Here we take the model above and add negative constraints. For example, on some transitions we add constraints like e.g. \( \text{not}(\text{iknows}(\text{Amount})) \). The part \( \text{iknows}(\text{Amount}) \) ensures that the intruder (i.e. the mediator) must know this data (the loan amount), and thus, \( \text{not}(\text{iknows}(\text{Amount})) \) ensures that this data must not be deductible in any way. In this situation, the Orchestrator do not finds any orchestration, which is logical since no Clerk is available and these constraints prevent the mediator to be a Clerk (i.e. the constraints do their job).

LOP model with neg. constraints and with Clerks Here we keep the model above and add the missing Clerk services. Now, the orchestrator generates a mediator which delegates some parts of the process to the Clerks. Analysis shows that the secret data remains secret and non-deductible by the mediator, thanks to the encryption layer added by the (real) Clerks, and that all goals are reached.

4.3 CORAS Tool

CORAS [47, 30] is a model-driven approach to risk analysis that consists of three tightly integrated building blocks, namely the CORAS method, the CORAS language and the CORAS tool.

When conducting a risk analysis using CORAS, diagrams made with the CORAS risk modeling language are used intensively during the structured brainstorming sessions of the risk analysis workshops for the purpose of risk identification, risk estimation, risk evaluation and risk treatment. Because models are made on-the-fly in these sessions, appropriate tool support is decisive for the success of an analysis; to avoid interrupting the flow of discussion, it is important that the models are made efficiently and are clearly presented. In the following we explain the basic features of the CORAS tool, referring to an example project. The example project is based on the risk assessment of patient monitoring services from the NESSoS eHealth use case. This risk assessment is described in D11.3 [32], but for the full documentation, the reader is referred to [48].

4.3.1 Features

The CORAS tool is a diagram editor supporting the modeling of risks and threats in the CORAS language. The CORAS method is based on the risk analysis process of the ISO 31000 standard [38] and provides tool and modeling support for all phases and activities of the process. Most importantly are asset diagrams for establishing the target of analysis, threat diagrams for risk identification, risk diagrams for risk evaluation, and treatment diagrams for the identification of options for risk mitigation.

The CORAS tool facilitates quick modeling and editing of syntactically correct diagrams by drag-and-drop and click-and-drag functionality, as well as automatic prevention or automatic alert of syntactically illegal expressions. Because the CORAS tool is a diagram editor for making CORAS diagrams it does not have input and output, other than the CORAS project files that are created.

Figure 4.3 shows a screenshot of the tool. The canvas in the middle is the drawing area for creating the diagrams. To the left is the palette which contains all the model elements and relations of the CORAS language. To the right is the outline presenting the current project and its diagrams in a tree structure. To the bottom is the properties window to display properties of selected elements, and to edit these properties.

4.3.2 Working with the CORAS Tool in SDE

In the following, we explain how the tool is used to support a CORAS risk analysis and which diagrams that need to be created to conduct the activities and to document the results. We have created an example project, named eHealth, with some small, initial diagrams to aid new users in getting started. The example project is one of the NESSoS application case studies as documented in Deliverable D11.2 [31], namely the patient monitoring scenario. The reader is referred to D11.2 and to Deliverable D10.3 [33] for details about this scenario. For further support on how to get started, the reader is also referred to the instruction video available online.8 This demo uses examples from Chapter 3 [46] of the CORAS book, which can be

8CORAS. http://coras.sourceforge.net/coras-tool-demo.htm
The CORAS process consists of eight steps that can be structured according to the five phases of the risk analysis process as defined by ISO 31000. We explain how the CORAS tool is used to support each of the phases in turn.

**Establish the context.** This phase comprises the first four steps of the CORAS process and includes identifying and documenting the stakeholders, describing the target of analysis, identifying the assets and defining the risk evaluation criteria. The stakeholders and assets are modeled and documented in the tool by means of **CORAS asset diagrams**. One diagram is created for each stakeholder and captures the assets of this stakeholder (given the target of analysis), and the relations between the assets.

Deliverable D11.2 [31] gives a detailed discussion of relevant security properties and assets in the eHealth domain. In our example project we have modeled four assets with respect to the stakeholder service provider, i.e. the provider of the patient monitoring services. These assets are **Electronic Health Record (EHR) security** (which includes confidentiality, integrity and availability), **patient privacy**, **compliance** with data protection laws and regulations, as well as **patient monitoring service provisioning**.

**Identify risks.** This corresponds to Step 5 of the CORAS process and is conducted using **CORAS threat diagrams**. The risk identification is conducted by identifying threats, vulnerabilities, threat scenarios and unwanted incidents with respect to the identified assets and given the documented target of analysis. The risk identification should be conducted by addressing different parts and aspects of the target of analysis in turn, making sure that all assets are covered. In the example project we have created two threat diagrams, one documenting risks related to the server side and one related to the monitored patient side.

For example, in Figure 4.3 **Malware causes leakage of EHRs** is an unwanted incident that harms the assets **EHR security** and **Privacy**. A risk is a pair of an unwanted incident and an asset that is harmed, so this particular incident constitutes two risks.

**Estimate risks.** This corresponds to Step 6 of the CORAS process and is conducted using the **treat diagrams** of the previous step as input. The objective is to estimate the likelihood of unwanted incidents and
their consequences for the assets they harm. To support the estimation of the likelihoods of the unwanted incidents, and to understand the most important sources of risks, likelihoods of threat scenarios should also be estimated. The estimates are modeled in the tool by annotating the relevant elements. In the example project we used the likelihoods unlikely, possible and certain, and the consequences insignificant, moderate, major and catastrophic. Each of these values must be precisely defined. Depending on the need, they may be qualitative or quantitative, and defined as intervals or using exact values. Likelihoods can be given in terms of probabilities or frequencies.

For example, in Figure 4.3, the threat scenario Server is infected by malware has likelihood unlikely. The probability $0.1$ on the relation to the unwanted incident Loss of integrity of EHRs is a conditional probability expressing that if the scenario occur there is a probability of $0.1$ that it will lead to the incident. The likelihood of the unwanted incident Malware causes leakage of EHRs is unlikely, and its consequences for EHR security and Privacy are major and catastrophic, respectively.

Evaluate risks. This corresponds to Step 7 of the CORAS process and is conducted using CORAS risk diagrams and with the threat diagrams of the previous steps as input. Using the CORAS risk diagrams, the likelihood and consequence estimates for each unwanted incident is modeled as a risk with its risk level. In the example project we used three risk levels, namely low, medium and high. For example, the combination of likelihood unlikely and consequence major yields risk level medium, whereas certain and moderate yields high.

The two risks constituted by the unwanted incident Malware causes leakage of EHRs have levels medium and high, respectively.

Treat risks. This corresponds to Step 8 of the CORAS process and is conducted using CORAS treatment diagrams, using the threat diagrams from Step 6 and the risk diagrams from Step 7 as input. The treatment identification is conducted by systematically going through the threat diagrams and focusing on the unacceptable risks. Proposed treatments are inserted in the treatment diagrams with relations to the elements they treat, such a threats or vulnerabilities. In the example project we also used CORAS treatment overview diagrams to give an overview of which of the identified risks are mitigated by which treatments.

4.4 EOS (Eye OCL Software)

EOS [29, 34] is a Java component for efficiently performing the evaluation of queries written in OCL [49] on medium-large size scenarios. OCL is the specification language chosen by model-driven security modeling languages like SecureUML [17] and MagicUWE [45, 23] for formalizing authorization constraints, which are conditions that constraint the permissions granting access to protected resources.

4.4.1 Features

The EOS component includes an OCL parser (which uses the parser generator SableCC\(^{10}\)) and an OCL evaluator, the latter consisting of about 7K lines of Java code. The current version handles most of OCL, including the possibility of adding user defined operations.

As reported in Deliverable D2.2 [25], EOS has been integrated into the SDE. The EOS/SDE plugin provides methods to insert elements, one-by-one, into ActionGUI data models (a subset of class diagrams) and scenarios (object diagrams), and to input OCL expressions for evaluation.

4.4.2 Working with EOS in SDE

As already mentioned, OCL is the specification language chosen by SecureUML [17] for formalizing dynamic, fine grained access control policies: i.e., policies that depend on the satisfaction of specific authorization constraints, where these constraints refer to the current values of the properties (either attributes or association-ends) of the resources which are trying to be accessed, and/or of the users who are trying to access these resources.

Interestingly, EOS can effectively be used to validate SecureUML models. More specifically, EOS can be used to check/test (at design time), over different scenarios that the authorization constraints declared

\(^{10}\)StableCC. http://sablecc.org/
Figure 4.4: Validating with EOS/SDE the ActionGUI security model for the EHealth Case Study

in a given SecureUML model will always grant or deny access (at run time) to the protected resources according to the intended security policy.

To illustrate this (security-related) usage of EOS within SDE, we have made available an example set at the SDE web page, containing a SDE script, named ehr.sscript that loads

- The ActionGUI data model for the EHealth Case Study, along with its prototypical scenario, as it has been reported in Deliverable D11.3 [32].
- Two OCL expressions that can be used to validate that only professionals of the same department can have read-access to the public information of a patient.
- Three OCL expressions that can be used to validate that only doctors of the same department can have read-access to the social information of a patient.

Notice that the OCL expressions included in the script ehr.sscript are concretizations, for different instances of users (caller) and different instances of patient public or social information (self) of the authorizations constraints

\[
\text{caller}.\text{worksIn} \rightarrow \text{includes}(\text{self}.\text{patient}.\text{department})
\]

and

\[
\text{caller}.\text{worksIn} \rightarrow \text{includes}(\text{self}.\text{patient}.\text{department}) \text{ and } \text{caller}.\text{myRole}.\text{role} = \text{Doctor}'
\]

which constraint the read-access to the public or social information of a patient, according to the ActionGUI security model for the EHealth Case Study, as it has reported in D 11.3. Figure 4.4 shows a screenshot of the EOS/SDE plugin after executing ehr.sscript.

Finally, to validate other authorization constraints included in the ActionGUI security model for the EHealth Case Study, or the validate them over a different scenario; simply change the script ehr.sscript accordingly. Remember to replace the variables self and caller by the appropriate resource and user instances.
4.5 Jalapa

Jalapa [14, 15] is an extension to the security model of Java that allows one to specify, analyze and enforce history-based usage policies. Programmers can sandbox an untrusted piece of code with a set of policies, which are monitored and enforced at run-time. The tool enables safe composition of programs or services with their own security requirements, in order to achieve call-by-contract service composition [16].

4.5.1 Features

This section first describes how policies for Jalapa look like. Second, the usage of Jalapa is briefly explained.

Policies

Policies are defined by the means of usage automata, a variant of Finite State Automata (FSA), where symbols in the input alphabet are parameterized over names (of objects). A graphic component for editing security policies is available in the version of Jalapa integrated in the SDE (see Figure 4.5).

Technically, Jalapa accepts as input a set of policies, chosen among those specified in an arbitrary number of files whose extension is .upy. An example policy, forbidding the execution of a write operation after a read is shown in the following listing. This example is specified by a simple classical automaton. However, more complex policies can take parameters in transition labels, e.g. read(x) or write(x). This allows one, for example, to check that a write does not happen after a read on the same file.

![Figure 4.5: Jalapa policy editor](image)

```java
name: chinese-wall
aliases:
read := (java.io.BufferedReader).readLine()
write := (java.io.BufferedWriter).write(String s, int off, int len)
states: q0 q1 fail
start: q0
```

NESSoS - 256980 42
Instrumentation

Jalapa can be either used as a stand-alone tool or integrated into the SDE. Detailed documentation, and several examples are available at the Jalapa website\(^1\).

Jalapa operates by instrumenting compiled Java classes, adding the necessary glue code to enable run-time monitoring, in the form of “proxy” classes that stand between the program and system libraries. The instrumentation procedure takes five arguments:

1. The input directory, containing classes to be instrumented.
2. The output directory, where instrumented classes will be written.
3. The application entry point.
4. The list of policies to be checked, according to the name field of a policy in a loaded .upy file.
5. The directory containing the policies to be loaded. All the files having a .upy extension in this directory are loaded and the policies defined therein can be checked.

Indeed, the obtained classes are not intended to be run in some special tool or environment: they are used in the way they are intended (be it in a program, or in an OSGI bundle, for example). The violation of a policy raises an exception of type jisel.policy.SecurityPolicyException at run time.

An example (untrusted) program that violates the chinese-wall policy is shown in the following:

```java
import java.io.*;
public class WriteFile {
    public static void main(String args[]) {
        try {
            BufferedReader rf = new BufferedReader(new FileReader(args[0]));
            BufferedWriter wf = new BufferedWriter(new FileWriter(args[1]));

            String s = rf.readLine();
            wf.write(s, 0, s.length());

            wf.close();
            rf.close();
        } catch (IOException e) {
        }
    }
}
```

4.5.2 Working with Jalapa in SDE

The instrumentation in Jalapa can be invoked from SDE. The relevant method is `instrument(input_dir, output_dir, entry_point, policies, policy_dir)`. The parameters, all of type String, are as specified in subsubsection 4.5.1. The first and second arguments are self-explanatory. The input directory is

\(^1\)Jalapa. [http://jalapa.sourceforge.net/](http://jalapa.sourceforge.net/)
assumed to contain pre-compiled java classes. The entry point is the name of the main class of the application. The policies to be checked are specified as a list of semicolon-separated policy names. The last parameter is a directory, assumed to contain a number of .upy files, where the selected policies are specified.

The first code listed above constitutes a valid .upy file. The second program (whose name will be WriteFile.java) can be compiled using the standard javac command, and then instrumented using the policy named chinese-wall, as shown in Figure 4.6, where cw is the name of the directory containing the compiled program. The result will be a compiled java class file. Monitoring can then be enabled at run time by loading jisel.jar (available in the Jalapa distribution), invoking the Java interpreter as follows:

```
java -Dcheck.global=chinese-wall -cp .:/path/to/jisel.jar WriteFile input output
```

where input and output actually are just arguments of the instrumented program, which can be found in the file WriteFile.class, in the output directory.

4.6 PRRS

The Platform for Runtime Re-configurability of Security (PRRS) is a tool that provides run-time management of Security and Dependability (S&D) solutions and monitoring of the system context.

4.6.1 Features

The features of PRRS are described in section 3.5. Furthermore that section describes how to execute the PRRS server. The running server is required to use the PRRS client which is described in the next section.
4.6.2 Working with PRRS in SDE

For this scenario, we have developed an encryption service that offers an Advanced Encryption Standard (AES) encryption solution implemented PRRS-aware. This client (a PRRS-aware application example) is integrated in SDE as a tool with a public interface called `encrypt`, with 2 parameters: `textPlain` and `password`.

When `encrypt` is invoked, PRRS searches the most suitable encryption solution in its databases (in our scenario, the `EncryptAESImplementation.xml`) and return an executable component (here, the `EncryptAES/EncryptAES.jar`) that performs the encryption and, in case of encryption fail, the PRRS server sends an event to our monitor pointing this fact. The encryption fails when a password with less than 8 characters is provided. However, in the client application the return value must be always `true`.

The examples from our example set (see Figure 4.7) are selected by pressing the “Load predefined SDE example set” button. The output is then shown in PRRS server terminal.

![Available examples for function “encrypt”](image)

![Invoke Function](image)

*Figure 4.7: Examples for the Encrypt function*

4.7 MagicUWE

The CASE tool MagicUWE [45] was created to support the development of Web applications focusing on the modeling phase.

4.7.1 Features

MagicUWE supports the UML-based Web Engineering (UWE) notation and the UML-based Web Engineering (UWE) development process, i.e. which comprises (1) extensions of the toolbar for comfortable use of UML-based Web Engineering (UWE) elements including shortcuts for some of them, (2) a specific menu to create UWE default packages and new diagrams for the different views of web applications (content, navigation, presentation and processes), and to execute model transformations, (3) additional context menus not only for the containment tree but also within navigation diagrams. In particular, MagicUWE supports modeling of security features of web applications [22]. For these security features specific stereotypes, tagged values and constraints were defined extending the UML-based Web Engineering (UWE) profile.
4.7.2 Working with MagicUWE in SDE

As MagicUWE is designed as a plugin for MagicDraw\(^2\) (v.16.8), a MagicDraw project is needed in order to execute `editProjectWithMagicUWE(File project)` as already shown in section 2.3.

The MagicUWE example project set, which can be downloaded from the SDE website\(^3\) contains several examples modeled with UWE. There are several versions of an address book to manage contacts, including one that uses UWE’s security features. Furthermore, a Hospital Information System (HIS) prototype called *HospInfo* is provided along with the UWE profile project (v.2.0) which is needed to make UWE stereotypes available.

Detailed information about the UWE projects can be found at the UWE website\(^4\).

4.8 UML4PF

The tool *UML4PF* supports requirements analysis according to an enhanced version of Jackson’s problem frame approach \(^5\). Problem frames are patterns classifying software development problems. They pay special attention to the environment in which the software (called *machine*) will operate. That environment is represented by means of a *context diagram* showing how the environment is structured in terms of *problem domains* and how the machine can interact with its environment. Interaction between domains is modeled by considering *shared phenomena*, which are controlled by only one domain and can be observed by other domains. Requirements are optative statements that refer to one or more problem domains and constrain at least one problem domain. Annotating (parts of) context diagrams with requirements yields *problem diagrams*. *Problem frames* are abstracted versions of problem diagrams. A simple subproblem of a more complex software development problem can be fitted to a problem frame by instantiating the frame diagram accordingly.

To provide tool support for frame-based problem analysis, we have carried over Jackson’s original notation to UML by defining a corresponding profile \(^6\). We developed an Eclipse-plugin for working with the defined profile. Furthermore, we developed a large number of *validation conditions* that make it possible to perform semantic checks on the developed diagrams. These validation conditions either refer to single diagrams (e.g., requirements are not allowed to constrain the machine), or they allow one to check the coherence between different diagrams (e.g., the messages of a sequence diagram must be phenomena of the corresponding problem diagram).

In this way, UML4PF supports software engineers in developing a coherent and complete set of requirements documents. Moreover, it supports the systematic development of an appropriate software architecture, see \(^7\). Next, we describe the technical realization of UML4PF. Then, we illustrate how to work with the tool.
4.8.1 Features

UML4PF\textsuperscript{15} consists of a UML [50] profile with formal validation conditions expressed in OCL [49] and an Eclipse plugin. In Figure 4.8, gray boxes denote re-used components, whereas white boxes describe those components that we created. The functionality of our tool comprises the following:

- The **UML Profile for Problem Frames** defines the relevant stereotypes for our approach, e.g., «ProblemDiagram».
- The **Requirements Editor** allows one to add new requirements.
- The **Model Generator** automatically generates model elements, e.g., it generates observed and controlled interfaces from association names.
- The **OCL Validator** checks if the model is valid and consistent by evaluating our OCL expressions. It also returns the location of invalid parts of the model. All in all, we have defined about 50 OCL validation conditions for the analysis phase. The time needed for checking only depends on EMF\textsuperscript{16} and is about 0.5 seconds per validation condition.
- The **sdgen Editor** is used to edit sequence diagrams.
- The **Interactive ModelTransformer** serves to create software architectures through interactive model transformations.

The tool UML4PF is still under development and evaluation.

4.8.2 Working with UML4PF in SDE

We first illustrate how to work with UML4PF and then give a hands-on introduction of how to use UML4PF within the SDE.

We consider a simple software development subproblem that is part of the larger task of developing an online vacation rentals system. Requirement R01 states that staff members make new holiday offers available, see Figure 4.9.

We create a new project with a new model both named *VacationRentals* and apply our UML profile to it using a EMF-compatible UML editor such as Papyrus\textsuperscript{17}.

**Problem elicitation and description:** We describe the intended environment of our vacation rentals system by a context diagram. This is achieved by using the graphical elements provided by Papyrus’ editor. Domains are represented as classes, and interfaces are represented as associations. The package is annotated with the stereotype «ContextDiagram». Next, we execute the model generator. After the generation, all interfaces corresponding to connections in the context diagram exist.

To check the consistency of the context diagram, we right-click on the UML-file, this time selecting the entry “Validate now”. The validation conditions are checked, and the results are displayed (see bottom of Figure 4.9). Fulfilled validation conditions are displayed in green, violated ones in red. For the violated conditions, we provide further expressions that indicate which elements cause the conditions to fail.

**Problem decomposition and classification:** We now decompose the overall problem into subproblems. For each subproblem, we create a package with the stereotype «ProblemDiagram». We re-use the classes of the context diagram where applicable. Furthermore, we add classes with the stereotype «Requirement» and assign the appropriate stereotype to the dependencies originating from this class (see Figure 4.9). UML4PF allows us to check if a described problem diagram is an instance of a pattern [37]. We also provide a figure of how UML4PF works within the SDE tool (see Figure 4.10).

**Derive Software Specification:** In this step, we draw sequence diagrams using the sdgen editor. For each problem diagram, we create sequence diagrams capturing normal and exceptional behavior. The domains in the problem diagram that are directly linked to the machine become lifelines in the respective sequence diagram. Operations in interfaces are mapped to messages.

\textsuperscript{13}MagicUWE example set. http://sde.pst.ifi.lmu.de/nessos/ExampleProjects/MagicUWEexamples.zip
\textsuperscript{14}UWE Website. http://uwe.pst.ifi.lmu.de/examples.html
\textsuperscript{15}UML4PF Website. http://uml4pf.org
\textsuperscript{16}Eclipse EMF. http://www.eclipse.org/emf/
\textsuperscript{17}Papyrus. http://www.papyrusuml.org
Figure 4.9: UML4PF: Screenshot of Problem Diagram for R01

Figure 4.10: OCL Validator inside the SDE platform
In the subsequent steps we:

- **derive the technical context diagram.** This is a special context diagram showing the technical details of the machine environment, e.g., the used web servers. We can check the consistency of context diagram and technical context diagram.

- **Specify operations and data structures.** The operations are specified using OCL expressions. The syntax of these expressions is checked via the built-in OCL parser. Furthermore, UML4PF can check for completeness of the operations against the sequence diagrams.

In the following we show a basic usage example of UML4PF in the SDE.

1. With the plugin installed you should be able to see a function overview of UML4PF.

2. The plugin needs an UML model to work on, which can be loaded by clicking on the function named `loadUML4PFModel`.

3. This function takes one string parameter, which is the path to the UML file (see Figure 4.11).

4. Click “ok” then “finish” and the model will be visible on the Blackboard.

5. Now you can call the `generateModelElements` function for auto-generation of required model elements. This function takes no parameter; It works on the object posted to the SDE Blackboard (see Figure 4.12).

6. Validation of the UML object can be done by calling the `validateUML4PFModel` function. This function takes no parameters; It works on the object posted to the Blackboard (see Figure 4.13).

7. All changes applied to the model during generation and validation can be rolled back by using the `revertChanges` function. This function takes no parameters, as it also works on the object posted to the Blackboard and saves it to the provided path in step 3 with all changes being rolled back.

![Invoke a tool function](image)

**Figure 4.11: loadUML4PFModel parameter**

### 4.9 VeriFast

One absolute truth about software is that programmers make mistakes during the development process. These software bugs can give rise to subtle problems, but may sometimes result in disastrous catastrophes. One example is the crash of the Ariane 5 rocket, where a simple integer overflow lead to the
destruction of this $370 million dollar machine. Another example is the Toyota recall due to a software bug in the car computer system. Some of these bugs are also so-called vulnerabilities. A vulnerability is a bug that can be exploited by an attacker to make the software do something it's not supposed to do. Most software contains such vulnerabilities, as demonstrated by Microsoft's monthly security updates.

4.9.1 Features

VeriFast [40] (partially) solves these problems by proving that software does not contain errors such as unallocated memory access, null pointer dereferences, out of bound errors, data races, and ensures compliance with API contracts. In addition, compliance with the application's own specifications can also be verified. However, these advantages do not come for free. VeriFast tries to reason about the application as much as it can, but some programmer interaction is required in order for VeriFast to be able to fully verify an application. Programmers must annotate their source code with method pre- and postconditions, invariants, mathematical datatype and function definitions, recursive memory structure definitions, inductive proofs, and potentially some proof steps. This annotated source code can then be processed by VeriFast, which will result in one of two outcomes. Either the program verifies, and then a mathematical proof exists that the application does not have any of the issues mentioned before. However, in the case that the program does not verify, an explanation of why the proof could not be constructed is given to the programmer.

The annotations can describe one of several building blocks that is used by the VeriFast approach. The following paragraphs give a short overview of these building blocks. More information can be found in [41, 42].

Method Contracts Developers can specify the behavior of a function via a method contract consisting of two assertions, a precondition and a postcondition. Both assertions are written in a form of separation logic. When VeriFast tries to verify a method, it will start by populating the symbolic state with everything
that is described in the precondition of the method. VeriFast will then symbolically execute the function and will verify that for each possible execution the postcondition of the method is reached.

**Inductive Data Types** VeriFast supports inductive data types to allow developers to specify rich properties. To describe recursive data structures and to allow for information hiding, VeriFast supports separation logic predicates. A predicate is a named assertion. Predicates can be precise, which means that their input parameters uniquely determine (1) the structure of the heap described by those predicates, and (2) the values of the output parameters. Input and output parameters are separated by a semicolon. VeriFast tries to automatically fold and unfold precise predicates whenever necessary. Developers can also insert explicit fold (close) and unfold (open) proof steps in the form of ghost commands for non-precise predicates or when the automatic (un)folding does not suffice.

**Fixpoint Functions** VeriFast also supports fixpoint functions. Just like predicates and inductive data types, fixpoint functions can only be mentioned in specifications, not in the source code itself. The body of a fixpoint function must be a switch statement over one of the fixpoint’s inductive arguments. To ensure soundness of the encoding of fixpoints, VeriFast checks that fixpoints terminate. In particular, VeriFast enforces that whenever a fixpoint $g$ is called in the body of a fixpoint $f$ that either $g$ appears before $f$ in the program text or that the call decreases the size of an inductive argument.

**Lemma Functions** Lemma functions allow developers to prove properties of their inductive data types, fixpoints and predicates, and allow them to use these properties when reasoning about programs. A lemma is a function without side-effects marked `lemma`. The contract of a lemma function corresponds to the property itself, its body to the proof and a lemma function call corresponds to an application of the property. VeriFast has two types of lemma functions: pure and spatial lemmas. A pure lemma is a function whose contract only contains pure assertions, and whose body proves that the precondition implies the postcondition. Spatial lemmas can mention spatial assertions such as predicates and points-to assertions. A spatial lemma with precondition $P$ and postcondition $Q$ states that the program state described by $P$ is equivalent to the state described by $Q$. A spatial lemma call does not modify the underlying values in the heap, but changes the symbolic representation of the program state.

### 4.9.2 Working with VeriFast in SDE

A feature that proved to be crucial in understanding failed verification attempts is VeriFast’s symbolic debugger, which is available in the NESSoS SDE. This debugger offers an interactive user interface that can be used to diagnose verification errors by inspecting the symbolic states encountered on the path to the error. For example, if the tool reports an array indexing error, one can look at the symbolic states to find out why the index is incorrect. This stands in stark contrast to most verification condition generation-based tools that simply report an error, but do not provide any help to understand the cause of the error.

VeriFast annotates functions with preconditions and postconditions written in separation logic. For example, consider the class `Interval` shown below. VeriFast will check that if the `shift` function is executed with the specified preconditions in place that the function reaches the postconditions. If VeriFast cannot prove this, either more annotations are required or the function has a bug in it.

```java
/*@ predicate interval(Interval i, int l, int h) =
 i.low |-> l &*& i.high |-> h &*& l <= h;
@*/

public class Interval {
    int low, high;

    void shift(int amount) {
        // @ requires interval(this, ?low, ?high);
        // @ ensures interval(this, low + amount, high + amount);
    }
}
```
Deliverable D8.3 [43] reports on the results that were obtained with VeriFast in the past year. In particular, a number of case studies have been performed on (relatively) large commercial software, and a large number of bugs have been detected as a result of this work.

4.10 X-CREATE

The tool X-CREATE allows for the automatic generation of XACML requests from a XACML policy. The current version of the X-CREATE tool integrated into the SDE implements the Simple Combinatorial testing strategy.

4.10.1 Features

In this testing strategy a combinatorial approach is applied to the policy values as explained in [20, 19]. We generate a simple request containing the values of the subject, resource, action and environment of the derived combination. The generated requests are obtained using combinatorial approaches implemented into the Combo-Test tool [10]. Specifically, they are first those obtained using the combinations of the Pairwise approach, then those ones using the combinations of the Threewise approach and finally those using the combinations of the Fourwise approach applied to the subjects, resources, actions and environments values of the policy. In this way, we try to generate a test suite guaranteeing a coverage first of all pairs, then of all triples and finally of all quadruples of subjects, resources, actions and environments values derived by the policy. The maximum number of requests derived by this strategy is equal to the cardinality of the Fourwise set. The main advantage of the proposed strategy is that it is simple and achieves the coverage of the policy input domain represented by the policy values combinations.

4.10.2 Working with X-CREATE in SDE

In this section we show how to use X-CREATE for deriving XACML requests from the demo-5 object-specific policy. By using multiple rules, this policy shows how to deny access to all raw datastreams in the object except to particular users (e.g., the object owners). It also shows how to deny access to a particular datastream to selected user roles.

We create a new project with demo-5 policy (see Figure 4.14) and we invoke the simpleCombinatorialStrategy function. Then we derive a set of XACML requests applying the Simple Combinatorial testing strategy described above. Specifically, for demo-5 policy, 84 XACML requests are generated. This number is the cardinality of the Threewise set of policy values. Figure 4.15 shows the set of derived requests. The requests appear in the SDE Blackboard in the order in which they are derived, using first the combinations of the Pairwise approach and then those of the Threewise approach. Note that, in this case the Fourwise approach is not applied because the environment values are missing in demo-5 policy. The derived requests, as shown in Figure 4.15, can be used as inputs for other SDE tools.

---

Figure 4.14: X-CREATE project

Figure 4.15: X-CREATE results
5 Towards Collaborations of Tools

The ability to compose new tools out of existing ones, is a process known as orchestration in the SOA world or in general as building tool chains. The objective of such a composition is to obtain a service that can be handled as a single tool. This means, the user can work with tool chains in the same way as with simple tools.

In this chapter, we present a first tool chain, comprising three tools, namely MagicUWE, UWE2XACML and XACML2FACPL. The connection of further tools, as e.g., X-Create to this tool chain is planned.

Naturally, the more tools are integrated into the SDE, the more possibilities for tool chains arise. Therefore, everybody is invited to integrate own security tools. LMU started to advertise the advantages of the SDE at conferences and workshops, e.g. ESORICS 2012\(^1\) (see also D12.4). Feedback of the attendants of these events was very positive.

5.1 Tool chains in the SDE

The SDE enables orchestration of tools via JavaScript or by using a SDE built in graphical editor, called SDE Orchestrator.

**Orchestrating with JavaScript.** The use of tool APIs directly within JavaScript enables developers to create a workflow by simply invoking tool functions and passing data in-between those functions. For enabling the newly created workflow to be usable as a tool in its own right, two requirements have to be fulfilled: (1) Instead of simply creating a workflow, a JavaScript function definition is required which states a function name and parameters. (2) As each tool, function, parameters, and return types may have descriptions and additional metadata attached, this metadata must be specified in the JavaScript source files. Both, 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.

**Graphical Orchestration.** Besides the ability to use JavaScript for orchestration as indicated above, the SDE also provides support to orchestrate tools graphically - a more intuitive manner to construct tool chains. 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 excerpt of a screenshot shown in Figure 5.1, which depicts the orchestration of the tools UWE2XACML and XACML2FACPL. For more details on this tool chain refer to the next section. Once modeled, an orchestration such as the one above is converted to a Java class, compiled in-memory and installed as a tool in the SDE.

To build tool chains of several tools, the output of one tool has to have the same format as the input of the subsequently connected tool. In case of different input and output formats, a tool for the transformations can be implemented to bridge the gap. For example, transformations between specifications of access control policies, as shown in the following section.

5.2 Example

In the second year of the project, we proposed the tool chain that enables the use of the tools MagicUWE, UWE2XACML and XACML2FACPL in one row. The resulting orchestration can be used as a service itself, which is called uweToFACPLTransformation. This includes modeling a secure web application and generating access control policies\(^4\) \[26\]. Tool chains can be created using the SDE Orchestrator, as shown in Figure 5.1.

---

\(^1\)ESORICS 2012 with a SDE flyer\(^2\), which was designed like the NESSoS\(^3\). [http://www.iit.cnr.it/esorics2012/](http://www.iit.cnr.it/esorics2012/)

\(^4\)UWE2FACPL. [http://uwe.pst.ifi.lmu.de/toolUWE2FACPL.html](http://uwe.pst.ifi.lmu.de/toolUWE2FACPL.html)
The *uweToFacplTransformation* service is started with a UWE project which provides a MagicDraw *.mdzip* file. After modeling access control policies with MagicUWE, it is exported in XMI format as input for the *UWE2XACML* transformation. The resulting XACML policy file is then transformed to be written in FACPL by *XACML2FACPL*. Our tool chain can be used exactly in the same way as other integrated tools.

In order to demonstrate the use of the *uweToFacplTransformation* tool we give an example from the health care domain. Our case study, called *HospInfo*, is a Hospital Information System. The roles identified for this web application are: visitor, registeredUser, nurse, receptionist, physician, and admin.

The main requirements of the *HospInfo* web-based system are:

- staff members should be able to register
- an administrator can set roles to staff members
- physicians need the permission to create new patient records or change information of patients
- nursing staff should be able to read the health records of the patients
- receptionists can read and update all information with exception of health related data, while only physicians can update the latter ones

The content model uses UML classes to represent the main concepts of the application being developed. The focus of interest is on the *Patient* class with attributes as name, address, ward or gender (see Figure 5.2). The classes *User* and *Role* (from the user model) are included as well in Figure 5.2, for showing the associations to the content model elements.

Figure 5.3 depicts the basic rights model of *HospInfo* with access specifications for the classes *User* and *Patient*. The rule that admins cannot change their own user account is depicted with the OCL [49] *authorizationConstraint* in the center of the diagram. Thereby, the variable *caller* stands for the operating

5 *HospInfo*. A secure hospital information system. [http://uwe.pst.ifi.lmu.de/exampleHospInfo.html](http://uwe.pst.ifi.lmu.de/exampleHospInfo.html)
user, i.e. the «updateAll» dependency between Receptionist and Patient specifies that the updates on all other attributes of Patient are permitted. Conversely, physicians can «updateAll» Patient attributes without any (except) restrictions.

Figure 5.4 shows an excerpt of the main navigation state diagram for HospInfo. The whole application HospInfo transmits all information in a confidential way and cares for the integrity and the freshness of the data (denoted by «session»(transmissionType="cif")). Basically, HospInfo consists of the two navigation areas depicted in Figure 5.4: a visitor area (on the left) and an internal area (on the right), which is guarded according to the existing roles. Below, we show the FACPL policies resulting from the transformation UWE2FACPL of the HospInfo basic rights model (only for roles receptionist and physician).

The transformation UWE2FACPL executes the methods UWE2XACML and XACML2FACPL, consequently it is part of our full tool chain depicted in Figure 5.1.

```xml
{permit-overrides ;
    target :{ string-equal("physician", subject.role) } ;
    (permit-overrides ;
        target :{ string-equal("patient", resource.id) } ;
        rules :
            (permit ;
                target :{ string-equal("update", action.id)
                u string-equal("name", resource.attr)
                - string-equal("birthYear", resource.attr)
                - ...
                - string-equal("ward", resource.attr) }
            )
            (deny) } i
    (permit-overrides ; target :{} ; rules :{(deny) } i )
}

{permit-overrides ; target :{string-equal("receptionist", subject.role)
    - string-equal("physician", subject.role));

    (permit-overrides ;
        target :{ string-equal("patient", resource.id) } ;
        rules :
            (permit ; target :{ string-equal("delete", action.id) })
    ) i }

...
Figure 5.3: *HospInfo* basic rights

Figure 5.4: *HospInfo* navigational states (excerpt)
6 Summary and Outlook

In this deliverable, we focus on the Service Development Environment (SDE), a service-oriented tool workbench that was selected as integration platform for security related tools within the scope of the NESSoS project.

Based on a service-oriented architecture itself, the SDE contains an increasing number of engineering tools for secure Future Internet (FI) software services and systems. The SDE not only provides tool information in a uniform way but also allows (remote) invocation of tool functionality and enables composition of tools by a textual as well as a graphical orchestration mechanism.

We present in this deliverable the improvements and extensions performed in the second year of the NESSoS project. A couple of extensions were implemented for the core of SDE: (1) Common Body of Knowledge (CBK) search and management of examples for integrated tools, (2) a set of updates were performed adapting to new versions of the underlying software platform, as well (3) several bugs were fixed.

New tools were integrated, such as UWE2XACML, X-CREATE, XACML2FACPL, AbsInt and PRRS. Tools which integration started during the first year of the project, such as Avantssar-atse (CL-ATSE), CORAS, Avantssar Orchestrator, Jalapa, MagicUWE, Eye OCL Software (EOS), UML4PF and Verifast, were completed and improved.

The SDE, including the integrated tools, is available for download at our dedicated tooling website, http://www.nessos-project.eu/sde. The website also contains a tutorial, a bug tracker and videos demonstrating the SDE in action. In addition, this deliverable contains a set of usage examples for the SDE, i.e. examples of how an integrated tool can be used within the tool workbench. An example of a first tool chain is presented as well. It comprises the use of three tools: MagicUWE, UWE2XACML and XACML2FACPL in a row.

The crucial point of course is to continue to enhance the integration of tools into the SDE from the project, from other European projects and from other interested groups. The tool workbench will be continuously updated.
7 NESSoS Second-Year Publications


Bibliography


NESSoS - 256980


