Network of Excellence

Deliverable D7.3

Prototypes supporting secure service architectures and design
<table>
<thead>
<tr>
<th>Project Number</th>
<th>256980</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Title</td>
<td>NESSoS</td>
</tr>
<tr>
<td>Deliverable Type</td>
<td>Report</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Deliverable Number</th>
<th>D7.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title of Deliverable</td>
<td>Prototypes supporting secure service architectures and design</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</td>
</tr>
<tr>
<td>Contractual Delivery Date</td>
<td>M24 September, 30th 2012</td>
</tr>
<tr>
<td>Actual Delivery Date</td>
<td>October, 15th 2012</td>
</tr>
<tr>
<td>Contributing WPs</td>
<td>WP 7</td>
</tr>
<tr>
<td>Editor(s)</td>
<td>Benoit Baudry, Olivier Bendavid</td>
</tr>
<tr>
<td>Author(s)</td>
<td>Benoit Baudry, Olivier Bendavid, INRIA</td>
</tr>
<tr>
<td></td>
<td>Marianne Busch, LMU</td>
</tr>
<tr>
<td></td>
<td>Nora Koch, LMU</td>
</tr>
<tr>
<td></td>
<td>Marina Soledad Egea Gonzalez, ATOS</td>
</tr>
<tr>
<td></td>
<td>Manuel Clavel, IMDEA</td>
</tr>
<tr>
<td></td>
<td>Miguel Angel Garcia de Dios, IMDEA</td>
</tr>
<tr>
<td></td>
<td>Riccardo Scandariato, KUL</td>
</tr>
<tr>
<td></td>
<td>Koen Yskout, KUL</td>
</tr>
<tr>
<td></td>
<td>Kristian Beckers, UDE</td>
</tr>
<tr>
<td></td>
<td>M. Carmen Fernandez Gago, UMA</td>
</tr>
<tr>
<td></td>
<td>Francisco Moyano, UMA</td>
</tr>
<tr>
<td>Reviewer(s)</td>
<td>Fabio Massacci- UNITN</td>
</tr>
<tr>
<td></td>
<td>Christoph Sprenger- ETH</td>
</tr>
<tr>
<td></td>
<td>Marinella Petrocchi - CNR</td>
</tr>
</tbody>
</table>
Abstract

The main objective of work package 7 is to provide improved support for architecting and designing secure services in Future Internet applications. This report presents seven prototypes, which all target specific modeling challenges for trust, access control and reputation in FI applications. This spectrum reflects the different concerns that impact the security of software architecture. It also reflects the challenges that arise in a context where applications have to run in constantly evolving environment.

Keyword List

Security, Future Internet, Design, Architecture, Model-based, Decomposition, Composition, Dynamic, Adaptation, Contract, Enforcement, Reusable Know-how
<table>
<thead>
<tr>
<th>Version</th>
<th>Type of Change</th>
<th>Author(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>initial setup</td>
<td>Benoit Baudry, Olivier Bendavid - INRIA</td>
</tr>
<tr>
<td>0.2</td>
<td>draft of intro-duction for ActionUWE: UWE to ActionGUI</td>
<td>Marianne Busch - LMU</td>
</tr>
<tr>
<td>0.3</td>
<td>initial draft</td>
<td>Marina Soledad Egea Gonzalez - ATOS; Manuel Clavel, Miguel Ángel García de Dios - IMDEA; -Benoit Baudry, Olivier Ben David - INRIA; Riccardo Scandariato, Koen Yskout - KUL; Marianne Busch, Nora Koch - LMU; Kristian Beckers - UDE; M. Carmen Fernandez Gago, Francisco Moyano - UMA.</td>
</tr>
<tr>
<td>1</td>
<td>Draft deliverable</td>
<td>B. Baudry - INRIA</td>
</tr>
<tr>
<td>1.1</td>
<td>Small improve-ments in the ActionUWE section</td>
<td>Marianne Busch - LMU</td>
</tr>
<tr>
<td>2</td>
<td>Under internal re-view</td>
<td>Massacci Fabio - UNITN, Sprenger Christoph - ETH, Petrocchi Marinella - CNR</td>
</tr>
<tr>
<td>2.8</td>
<td>After reviews</td>
<td>Marina Soledad Egea Gonzalez - ATOS; Manuel Clavel, Miguel Ángel García de Dios - IMDEA; -Benoit Baudry, Olivier Ben David - INRIA; Riccardo Scandariato, Koen Yskout - KUL; Marianne Busch, Nora Koch - LMU; Kristian Beckers - UDE; M. Carmen Fernandez Gago, Francisco Moyano - UMA.</td>
</tr>
<tr>
<td>2.9</td>
<td>Almost final</td>
<td>Marina Soledad Egea Gonzalez - ATOS; Manuel Clavel, Miguel Ángel García de Dios - IMDEA; -Benoit Baudry, Olivier Ben David - INRIA; Riccardo Scandariato, Koen Yskout - KUL; Marianne Busch, Nora Koch - LMU; Kristian Beckers - UDE; M. Carmen Fernandez Gago, Francisco Moyano - UMA.</td>
</tr>
<tr>
<td>3</td>
<td>Approved by the internal reviewers</td>
<td>Marina Soledad Egea Gonzalez - ATOS; Manuel Clavel, Miguel Ángel García de Dios - IMDEA; -Benoit Baudry, Olivier Ben David - INRIA; Riccardo Scandariato, Koen Yskout - KUL; Marianne Busch, Nora Koch - LMU; Kristian Beckers - UDE; M. Carmen Fernandez Gago, Francisco Moyano - UMA.</td>
</tr>
</tbody>
</table>
# Table of Contents

**ACRONYMS** ........................................................................................................ 9

**LIST OF FIGURES** .......................................................................................... 11

**LIST OF TABLES** .............................................................................................. 13

1 **INTRODUCTION** ......................................................................................... 15
   1.1 Integration activities ................................................................. 16

2 **A TRUST AND REPUTATION DEVELOPMENT FRAMEWORK** ............... 17
   2.1 Background ........................................................................ 17
   2.2 Problem Statement .............................................................. 17
   2.3 Approach ........................................................................... 18
   2.4 Prototype...... ...................................................................... 18
      2.4.1 Framework Requirements and Architecture ...................... 19

3 **EXPERIMENTAL PLATFORM** .................................................................... 23
   3.1 Background ........................................................................ 23
   3.2 Problem statement ............................................................... 23
   3.3 Approach ........................................................................... 23
   3.4 Prototype........................................................................... 24
      3.4.1 Experiment.................................................................. 24
      3.4.2 Tool .......................................................................... 24
      3.4.3 Towards a general experimental platform ....................... 28

4 **PATTERN-BASED CLOUD ANALYSIS** ....................................................... 29
   4.1 Background about the ISO 27001 Standard .............................. 29
   4.2 The UML4PF - Cloud Pattern Analysis Prototype......................... 31
   4.3 Implementation Overview...................................................... 34

5 **COMBINING CHANGE PATTERNS AND RUNTIME ADAPTATION** .............. 37
   5.1 Background ........................................................................ 37
   5.2 Problem statement ............................................................... 37
   5.3 Approach ........................................................................... 37
   5.4 Prototype........................................................................... 38
      5.4.1 Running example .......................................................... 38
      5.4.2 Stakeholders and supported tasks ................................. 38
      5.4.3 Under the hood ............................................................... 43

6 **ACCESS CONTROL ON PERSUSIVE ARCHITECTURES** ......................... 45
   6.1 Background ........................................................................ 45
      6.1.1 Models@runtime paradigm ........................................... 45
      6.1.2 RBAC (Role-based access-control) ................................. 45
      6.1.3 Kevoee ................................................................. 45
<table>
<thead>
<tr>
<th>Acronyms</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DoW</td>
<td>description of work</td>
</tr>
<tr>
<td>WP</td>
<td>work package</td>
</tr>
<tr>
<td>NESSOS</td>
<td>Network of Excellence on Engineering Secure Future Internet Software Services and Systems</td>
</tr>
<tr>
<td>CBK</td>
<td>Common Body of Knowledge</td>
</tr>
<tr>
<td>INRIA</td>
<td>Institut National de Recherche en Informatique et Automatique</td>
</tr>
<tr>
<td>SBT</td>
<td>self booking tool</td>
</tr>
<tr>
<td>RIA</td>
<td>rich internet applications</td>
</tr>
<tr>
<td>TCSEC</td>
<td>trust computer system evaluation criteria</td>
</tr>
<tr>
<td>DAC</td>
<td>discretionary access control</td>
</tr>
<tr>
<td>MAC</td>
<td>mandatory access control</td>
</tr>
<tr>
<td>RBAC</td>
<td>role based access-control</td>
</tr>
<tr>
<td>NPD</td>
<td>name protection domain</td>
</tr>
<tr>
<td>ACL</td>
<td>access control list</td>
</tr>
<tr>
<td>POC</td>
<td>proof of concept</td>
</tr>
<tr>
<td>DOM</td>
<td>document object model</td>
</tr>
<tr>
<td>JNA</td>
<td>Java Native Access</td>
</tr>
<tr>
<td>JML</td>
<td>java modeling language</td>
</tr>
<tr>
<td>RPC</td>
<td>remote procedure call</td>
</tr>
<tr>
<td>GWT</td>
<td>google web toolkit</td>
</tr>
<tr>
<td>JDK</td>
<td>java development kit</td>
</tr>
<tr>
<td>JRE</td>
<td>java runtime environment</td>
</tr>
<tr>
<td>SDK</td>
<td>software development kit</td>
</tr>
<tr>
<td>AVD</td>
<td>android virtual device</td>
</tr>
<tr>
<td>ADT</td>
<td>android development tools</td>
</tr>
<tr>
<td>SAML</td>
<td>Security assertion markup language</td>
</tr>
<tr>
<td>XACML</td>
<td>eXtensible access-control markup language</td>
</tr>
<tr>
<td>PEP</td>
<td>policy enforcement point</td>
</tr>
<tr>
<td>PDP</td>
<td>policy decision point</td>
</tr>
<tr>
<td>PIP</td>
<td>policy information point</td>
</tr>
<tr>
<td>PAP</td>
<td>policy administration point</td>
</tr>
<tr>
<td>TAM</td>
<td>target architectural model</td>
</tr>
<tr>
<td>RAM</td>
<td>reflecting architectural model</td>
</tr>
<tr>
<td>ACR</td>
<td>access-control reasoner</td>
</tr>
</tbody>
</table>
# List of Figures

Figure 1.1: Modelling secure architectures: three dimensions explored in WP7 contributions.  
Figure 2.1: Trust Conceptual Model (i).  
Figure 2.2: Trust Conceptual Model (ii).  
Figure 2.3: Trust Conceptual Model (iii).  
Figure 2.4: Framework Architecture.  
Figure 3.1: Partial screenshot of the tool, configured for the group with annotations, showing the catalog browser on the left and the process wizard on the right.  
Figure 3.2: Partial screenshot of the tool, configured for the group without annotations.  
Figure 4.1: Cloud System Analysis Pattern.  
Figure 4.2: Cloud System Analysis Pattern with Phone Support.  
Figure 4.3: Cloud System Analysis Pattern Validation Functionality.  
Figure 4.4: Cloud System Analysis Pattern Architecture.  
Figure 4.5: Cloud System Analysis Pattern EMF Model.  
Figure 5.1: Problem frames models of the example.  
Figure 5.2: Initial (top) and final (bottom) architecture of the eHealth example in Kevoree.  
Figure 5.3: Change Patterns Dashboard.  
Figure 5.4: Change pattern instance details.  
Figure 5.5: Preparing a solution by specifying the binding.  
Figure 5.6: Architecture of the prototype.  
Figure 6.1: Approach overview.  
Figure 7.1: Overview of ActionUWE Transformation.  
Figure 8.1: ATL2FOL prototype.  
Figure 8.2: ATL2FOL prototype activity.  
Figure 8.3: ER and REL metamodels.
Figure 8.4: The ATL transformation ER2REL ......................................................... 56
Figure 8.5: RBAC extension for the metaclass schema ............................................. 57
Figure 8.6: Extension of ER2REL to map RBAC structures .................................... 58
List of Tables

Table 2.1: Synthetic positioning of a trust and reputation framework ........................................... 17

Table 3.1: Synthetic positioning of an experimental Platform ..................................................... 23

Table 4.1: Synthetic positioning of a Cloud Pattern Analysis Tool .............................................. 29

Table 4.2: ISO 27001 [27] Section 4 .......................................................................................... 30

Table 5.1: Synthetic positioning of change patterns for runtime adaptation prototype .................. 37

Table 6.1: Synthetic positioning of the approach ........................................................................... 45

Table 7.1: Synthetic positioning of ActionUWE ........................................................................... 49

Table 8.1: Synthetic positioning of the verification of declarative ATL transformations .............. 53
1 Introduction

NESSoS WP7 focuses on methods and tools to model secure and adaptive architectures for software applications deployed on the Future Internet [26]. A major characteristic of Future Internet applications is "the fading boundary between development time and runtime" [22]. Consequently, the system architecture is designed in the early development phases, but then constantly evolves through the whole development lifecycle, including in post-deployment phases. This is reflected in the contributions integrated in WP7, which propose model-based concepts to integrate different security concerns in software architecture through the whole system lifecycle.

The workpackage contributions can thus be organized according to three dimensions: the security concern modeled and analyzed by a given technique; the phase in the lifecycle at which a given technique proposes to model and analyze security concerns; the formalism used to model security and the system. Figure 1.1 represents the different solutions the workpackage's contributions investigate in each dimension. The tools and prototypes in WP7 support security analysis according to trust, access control or security standards for information systems. These security concerns are modeled in early design models, detailed analysis models and for runtime deployment, using one or a combination of the following formalisms: UML, Secure UML, ADL (an architecture description language) or OCL.

Figure 1.1: Modelling secure architectures: three dimensions explored in WP7 contributions

This deliverable focuses on the presentation of tools and prototypes that support the modeling methods and techniques integrated in WP7. The position of each prototype with respect to the dimensions of figure 1.1 is explicitly stated at the beginning of each section, in a table that summarizes the following information:

- **purpose of the prototype**: this summarizes the main modeling intention that the prototype can support, as well as the main functionality the prototype proposes.
- **modeling formalism**: prototypes use general purpose formalisms such as UML (sometimes extended with profiles), OCL, Ecore, as well as specific languages for architecture description (ADLs) or model transformation (ATL)
- **security concern**: at the architecture level, the different prototypes address access control issues (in web navigation, pervasive systems and information systems), trust and reputation issues or the adaptation of standard information systems security concerns to cloud infrastructures.
foreground / background: each prototype leverages knowledge and techniques that preexisted the NESSOS NoE. Here, the authors clarify the position of foreground and background material that is included in the prototype.

1.1 Integration activities

The development of prototypes to support model-based secure software architecture, fosters the integration efforts of the network. In particular, this deliverable presents the following prototypes developed by two or more partners:

- ETH, IMDEA and LMU integrate their prototypes for model-based development of secure web applications in a novel prototype called ActionUWE (section 7)
- ATOS and INRIA integrate their expertise in formal logics modeling and model transformation in a prototype that aims at verifying the preservation of security properties through automatic model transformation (section 8)
- KUL and INRIA have initiated an integration effort to combine their respective prototypes on (i) the translation of security requirements changes into architecture patterns and (ii) dynamic system reconfiguration according to changes in the architecture (section 5)
- UDE and UMA have plans to deploy their prototypes (sections 2 and 4) over the Kevoree\(^1\) component-based runtime infrastructure developed at INRIA.

\(^{1}\)www.kevoree.org
2 A Trust and Reputation Development Framework

<table>
<thead>
<tr>
<th>Purpose of the prototype</th>
<th>Building trust and reputation models as integral parts of services and applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modeling formalism</td>
<td>Java, Kevoree in Y3</td>
</tr>
<tr>
<td>Security concern</td>
<td>Trust and reputation</td>
</tr>
<tr>
<td>Foreground / background</td>
<td>Everything is being designed specifically for NESSoS</td>
</tr>
</tbody>
</table>
| Table 2.1: Synthetic positioning of a trust and reputation framework

2.1 Background

The upcoming establishment of FI scenarios in our daily life requires the provisioning of trust between entities taking part in these scenarios. The shift to FI causes an important increase in the number of stakeholders and entities. In most cases, the realization of FI scenarios depend on the collaboration between these entities. However, due to the huge amount of entities, we cannot expect that they know each other beforehand, therefore they must decide at runtime whether they are going to collaborate or not. Here it is that trust comes into play, since it can be a powerful mechanism to leverage decision-making processes. Each entity wanting to interact with another entity should implement a trust model, which decides whether the entity trusts enough the other entity to proceed with the interaction.

Compared to trust, reputation is more objective and as stated in [31], constitutes a mechanism to build trust. Reputation models gather opinions about one entity from the rest of the population and assign a reputation score to this entity based on these opinions. Then, this reputation score is made public and is accessed by the entities in the system, which might incorporate it in their trust model in order to make a more informed decision.

The concept and implications of trust are embodied in the so-called trust models, which define the rules to process trust in an automatic or semi-automatic way in a computational setting. There are different types of trust models, each one considering trust in different ways and for different purposes.

Two seminal contributions were the ones by Marsh [35] and Blaze [6], and they already revealed the two main branches or categories of trust models that have been followed until today, and which we classified in evaluation models and decision models [40].

Trust management has its origins in decision models, the aim of which was to make more flexible access control decisions, simplifying the traditional two-step access control process (i.e. first authentication, then authorization) into a one-step trust decision. In these models, the system checks if a set of credentials attached to an entity match a policy, specified by a policy language.

On the other hand, evaluation models aim to assess to what extent an entity trusts another entity according to several criteria, such as previous interactions. The focus in these models is put on the evaluation process, which requires to identify and model the factors that have an influence on trust and aggregate them into a trust score.

2.2 Problem Statement

One issue with trust models is that they are very context-dependent, and are often designed as ad-hoc mechanisms to work within a given application. Actually, the standard is to plug a trust model into an existing, already-built application after-the-fact. This might lead to architectural mismatches between the application and the model, and the reusability of the model could also be reduced. Moreover, it is not possible for the model to exploit all the information available to the application, since there is no systematic procedure to include the model as a holistic part of the application. As a consequence, there are no methods to consider trust requirements from the very beginning of the software development lifecycle or to align the design of the model with the design of the application.
2.3 Approach

In order to overcome these limitations, we propose an object-oriented development framework that allows implementing trust evaluation models as a core part of the applications themselves. Thus, the purpose of the prototype is to assist developers during the development of different kind of applications that might require using evaluation models.

2.4 Prototype

For this purpose, we first performed a domain analysis of trust models [40], and as a result, we elaborated a conceptual model for trust, which is depicted in Figure 2.1, 2.2 and 2.3.

This domain analysis is very important for building the framework, since it highlights the most important concepts that are often present in trust models. On the one hand, it helps during the elicitation of the requirements that the framework should fulfill. On the other hand, it is a valuable input to the framework architecture design, since it sheds light on potential mappings from domain concepts to object-oriented concepts. These issues, deeply discussed in [39], are further developed in the following section.
2.4.1 Framework Requirements and Architecture

The framework must fulfill several requirements. From a high-level view, the framework must support the implementation of three types of evaluation models, namely reputation, behavior and propagation models (see Figure 2.1). Unlike the former, which basically compute an objective, reputation score for an entity, both behavior and propagation models rely on trust relationships between entities. The difference between these two models is that behavior ones set up new trust relationships according to the behavior of entities, whereas propagation models use existing trust relationships to derive new ones (e.g. by using transitivity).

The ultimate goal of the framework is to allow developers to implement both existing evaluation models and new ones. The following list of requirements describes the coarse-grained functionality that the framework should provide to developers:

- Entities management: trust is a relation between entities. The framework must allow the creation, binding and naming of entities.

- Trust relationships management: trust relationships might change over time. New trust relationships might be created (e.g. by propagation models), other relationships might be deleted, and it is likely that trust values change as well.

- Trust metrics definition: although the framework can provide some default built-in metrics implementations, it is important to let developers to define their own trust metrics, as they are the core concept in evaluation models.

- Variables management: variables are the atoms of metrics. It is important to let developers to create new variables, which can be used by user-defined metrics.

- Computation engines management: an engine implements a trust metric. This engine uses variables according to certain rules. Engines range from simple summation or average functions to complex fuzzy and probability distributions.

- Indirect trust computation: the framework should provide ways to determine the value of an undefined trust relationship based on defined ones by propagating trust information.

- Operators definition: indirect trust computation relies on operators that take trust paths as input and return trust values as output (and thus, a new trust relationship). Although several operators should be provided by default, the framework should allow developers to define new operators.
Regarding the architecture supporting these requirements, its structural view is depicted as a class diagram in Figure 2.4. Note that some classes have been mapped directly from the conceptual model described above, such as Entity and TrustRelationship among others.

The architecture follows a layered design, where each layer uses the services provided by the lower layer. A brief description of each layer is given next:

- **Model Layer**: this layer encapsulates the classes that manage the evaluation models to be implemented. They all share a context (a string describing the context under which the model operates). In the case of reputation models, there is a connector to external database systems to store the reputation of other entities in the system. A behavior model contains a list of trust relationships and exposes methods to get and set these relationships. Finally, a propagation model, in addition to containing a list of trust relationships, it also contains a sequential operator and a parallel operator.

- **Relational Layer**: this layer contains the basic building blocks onto which the models of the upper layer are developed: entities and trust relationships. Entities have a name, an automatically-generated identifier, a database connector and a trust metric. Regarding trust relationships, they consist of a tuple that specifies which is the entity that places trust (trustor), the entity on which trust is placed (trustee), the extent to which the trustor trusts the trustee (value), and the trust metric used to derive this value. Having the metric as an instance variable in these classes improves flexibility as each entity and trust relationship could be measured with different metrics.
• Computation Layer: evaluation models heavily rely on trust metrics to perform trust values calculations. This is the layer in charge of such computation. Basically, TrustMetric is an interface that a developer should implement to override the compute() method, where the trust calculation takes place. Trust metrics use variables, through the class Variable, which have a name and a value, as well as methods to get and set these parameters. Operators for propagation models belong also to this layer. Note that trust metrics contain instances of variables. As entities and trust relationships hold in turn instances of trust metrics, each entity or relationship might use different variables, increasing the flexibility of the framework to accommodate complex models.

• User-Defined Layer: this layer is created as users extend the computation layer to accommodate their own definitions. As we explain in the next section, users can create new computation engines (implementations of the TrustMetric interface) and new variables to implement an important range of models. For illustration purposes, the architecture includes a summation engine (that basically sums up the variables that it contains) and a weighted summation engine (that adds a weight to each variable). The latter requires creating a specialized variable class that adds the weight to its internal state.

The framework follows a grey-box approach, where the developer can use several functionalities in a black-box fashion as well as define new functionalities based on his needs. Given that the framework is to be implemented in Java, the developer will use this language as the formalism to use the prototype
3 Experimental Platform

<table>
<thead>
<tr>
<th>Purpose of the prototype</th>
<th>Performing empirical studies concerning software design processes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modeling formalism</td>
<td>UML, Ecore</td>
</tr>
<tr>
<td>Security concern</td>
<td>Security patterns</td>
</tr>
<tr>
<td>Foreground / background</td>
<td>The prototype was developed entirely in the NESSoS project, based on an existing security pattern catalog and the Topcased UML modeler</td>
</tr>
</tbody>
</table>

Table 3.1: Synthetic positioning of an experimental Platform

Controlled experiments are a useful tool to objectively evaluate a proposed technique, tool or process. When attempting to perform this kind of experiment for software design processes and techniques, however, practical difficulties emerge. For example, the participants of the experiment may not strictly adhere to the process they are supposed to follow, or they may misreport the time they have spent on their task. We present a prototype that has been developed to perform an empirical study in the domain of software design with security patterns, that attempts to overcome these problems.

3.1 Background

Controlled experiments can be used to objectively and empirically evaluate some technique, tool or process, by comparing it with an alternative technique and assessing the differences or similarities. The principles for conducting such controlled experiments are well-established. For instance, to assess the performance benefit of a new tool, two groups of participants (one with and one without the tool, usually balanced by factors such as experience, background, etc.) attempt to perform the same (or similar) tasks within a controlled setting that tries to eliminate the effect of any factor other than the used tool. A measure of the participants performance (time, for example) is compared between the two groups, possibly leading to a statistical significant difference.

3.2 Problem statement

When attempting to perform this kind of experiment to evaluate the benefits of software design processes and techniques, however, practical difficulties emerge. For instance, the participants of the experiment may not strictly adhere to the process they are supposed to follow, omit certain activities or switch the order of activities, making it difficult or meaningless to compare the obtained data. As another example, it is difficult to keep track of the time the participants have effectively spent, because software design is a complex and long-running task. Relying on the time that is reported by the participants themselves often leads to inaccuracies. This problem becomes worse when more fine-grained data is required (for example, the time per activity prescribed by the process). Better data may be obtained if the researcher can closely monitor the participant throughout the experiment, but this approach does not scale, especially with the large number of participants and long-running tasks that are often necessary in order to achieve statistically significant results.

The question arises whether scalable, automated data collection is possible for controlled experiments about (secure) software design.

3.3 Approach

To perform an empirical study in the domain of software design with security patterns [48], we have attempted to overcome these problems by means of tool-supported data collection. This chapter shortly describes the experiment for which the tool was developed, the functionality of the tool itself, and a discussion on what is needed to transform the tool into a general experimental platform for software design experiments. At the same time, the description of the tool highlights some attention points that may be taken into account by others when implementing a similar tool.
3.4 Prototype

Before describing the functionality of the tool itself, this section shortly describes the experiment for which it was developed.

3.4.1 Experiment

The experiment focuses on security patterns for software architecture design as well as detailed design. A security pattern describes a particular recurring security problem that arises in specific contexts and presents a generic scheme for its solution [44]. The description of a pattern follows the familiar template of, for instance, the Gang of Four patterns, including sections on the problem, the forces, the solution, the known uses, and so on. Security patterns can be a valuable vehicle to design secure software, as they provide sound, time-proven solutions. An inventory done in 2007 counted over 220 security patterns that had been documented over the previous ten years [25]. While there is clearly no shortage of security patterns in the literature, they are not nearly as popular as the software design patterns (e.g., Gang of Four). An investigation by Laverdiere et al. [34] has discovered that security patterns are plagued by a number of issues, including over- and under-specification, lack of generality and consensus, as well as misrepresentation. The mass of security patterns has also led to overlaps, as reported by Hafiz et al. [24]. These are all potential setbacks for a wider adoption.

The secure software engineering community is trying to mitigate the present situation by focusing on the rationalization of the patterns landscape, in order to make the offering more accessible, structured, and effective. In this stream, most activities have focused on (1) classifying patterns, e.g., via taxonomies and (2) provide support for navigation among patterns that are related, e.g., via pattern languages. These proposals result in extra tags and annotations to be included in the pattern documentation in order to improve usability [23]. However, to date, no evidence has been collected that any of these annotations do provide an advantage to the end users, i.e., architects and software designers.

The key research question of the experiment, then, is whether security patterns annotations increase the performance of an architect when making architectural choices. Here, an architectural choice amounts to the evaluation and selection of a solution in the form of a security pattern.

For the study, we started from a catalog of 35 security patterns, enriched with 4 annotation types [43]. The annotations provide extra information about the applicability of each pattern, its relationships with high-level security goals and other patterns, and its impact on other software qualities.

In summary, the study divided 45 teams of master students in two treatment groups. The students had to perform 4 design tasks involving the hardening of a software architecture via security patterns. One treatment group had access to a version of the catalog that is augmented with the above-mentioned annotations. The other group had access to the plain pattern documentation only. We have measured the performance of the teams in terms of (1) the time it takes to carry out each task (as an approximation of effort) and (2) efficiency, i.e., the number of patterns that are browsed in order to come to a solution. The results suggest that there is no significant difference between the two groups in terms of time. However, the group using the annotated catalog is more efficient. To explain this phenomenon, we observed that the group with annotations appeared to be more focused when choosing a suitable pattern, as they invested their time in gaining a deeper insight into a subset of the patterns. This could be the result of filtering the catalog based on the security objective annotations. Also, it is plausible that the presence of relationship information led that group to select additional (complementary) patterns, thereby increasing the time they spent selecting patterns.

3.4.2 Tool

The experiment was conducted using a tool that was used independently by the participants, both during (organized and supervised) lab sessions as well as (unsupervised) at home. The tool fulfills four purposes: (1) presentation of the tasks in the right order, (2) enforcement of the process that has to be followed, (3) providing the pattern catalog, with or without annotations depending on the treatment, and (4) the collection and submission of the traces and measurements.

Each team receives a unique code, which is entered when first starting the tool. The tool then configures itself for that specific team, for example by loading the correct order of the tasks and enabling or
Figure 3.1: Partial screenshot of the tool, configured for the group with annotations, showing the catalog browser on the left and the process wizard on the right

disabling the display of the annotations in the catalog browser, depending on the team’s assignment to the treatment groups. The configuration of the tool is done via an (encrypted) configuration file that contains, for all teams, the unique code, the treatment group assignment, and the task sequence. In this way, only one version of the tool had to be distributed to all participants.

The most important difference with respect to the tool’s user interface for both groups is the presence or absence of annotations in the pattern descriptions. The screenshots in Figures 3.1 and 3.2 show the tool’s interface for each treatment group. The tool is a customized Eclipse environment, consisting of a viewer to display the description of the tasks to be performed, a browser to navigate the security pattern catalog (left-hand side in Figure 3.1), a wizard to enforce the process described above (right-hand side), and the Topcased UML Editor\(^1\) (not shown in the screenshots).

**Task presentation** The tasks are presented to the teams only via the tool (i.e., not on paper or online) to make it harder to pass the task descriptions around (which would allow teams to already take subsequent tasks into account, thereby biasing the time measures), and to enforce the task execution order. The first task is a warm-up task in order for the team to become familiar with the system, the tool and the pattern catalog. Each team performs this task first. The other four tasks have to be completed sequentially (i.e., postponing a task is not allowed), and incrementally (i.e., each task is executed in the context of the architecture resulting from the previous tasks). In order to weed out the learning effect, the order of execution of tasks B–E is randomized across the teams.

In practice, the tasks are described in the form of HTML documents, which are displayed by a browser which is part of the tool. The files that contain these HTML documents are encrypted, to prevent the

\(^1\)Version 4.3.0, available from [http://www.topcased.org](http://www.topcased.org)
Figure 3.2: Partial screenshot of the tool, configured for the group without annotations

participants from (easily) gaining access to them outside the tool environment.

**Process enforcement**  All teams were given a process to follow when hardening the architecture. This provides the students with the necessary guidance so that they do not get "stuck" with the tasks, as they never used security patterns before. The process is lightweight and very intuitive, as it resembles any problem solving approach: a list of potential solutions is scouted first and then the most adequate is chosen. Furthermore, and more importantly for the experiment, strict adherence to this process allowed the experimenters to obtain detailed observations regarding the manner in which patterns were selected. For each task, the teams follow the steps below. Note that the process wizard in the tool enforces the sequential execution of the tasks, and of the process steps within each task.

1. **Study** the requirement that should be implemented and assess which parts of the architecture are impacted and how. If the requirement is already sufficiently supported by the current architecture, skip all remaining steps and go to the next task. The tool requires the team to enter a description of the expected impact, or to skip the remaining steps after giving a rationale for their choice.

2. Quickly skim through the security pattern catalog, and create a **shortlist** of possibly interesting patterns for the current task. The tool provides the security pattern catalog (recording each accessed pattern), and keeps track of the created shortlist.

3. Study the patterns from the shortlist more thoroughly, evaluate the real effectiveness of each candidate, make trade-offs with other qualities that are important for the architecture, detect conflicts with already implemented patterns and make a **final selection** of patterns to instantiate. The tool again displays the security pattern catalog, and keeps track of the final selection.

4. **Instantiate** the solution in the architecture by injecting the pattern in the design. This step is not used in the context of this experiment. In this step, the tool is used to display the security pattern
catalog. Also, the read-only protection on the UML editor is lifted during this step of the process, which ensures that all modeling activities happen exclusively during this step.

When no suitable pattern is found in steps 2 or 3, it is allowed to go directly to step 4 and implement a custom solution instead. In this case, however, the team is required to explain why none of the patterns suits their needs and the measurement is discarded from the data set.

For the teams in the group with annotations, the process steps are the same, but the activities are augmented so that the students can take advantage of the annotations in the catalog they are using. In particular:

- In step 1, the team must assign a security objective to the requirement.
- In step 2, the security objective is to be used to quickly select the relevant patterns for the shortlist. Further, the annotations about the relationships with other patterns can be used to extend the shortlist.
- In step 3, the annotations about the conflicts with other patterns and the trade-offs (with performance, availability and modifiability) are to be used to facilitate the final selection.

In summary, besides the order of the process steps and the display of the pattern catalog, the tool enforced the following aspects of the process. During step 1 (study of the requirement), the team has to use the wizard to enter its considerations about the impact of the requirement and, for the group with annotations, the security objective that has been assigned to the requirement. In steps 2 (shortlisting) and 3 (selection), the wizard must be used to enter the shortlist and the final selection. During steps 1–3, the diagrams of the architecture are accessible in read-only mode via the UML editor. The UML tab is enabled for editing during step 4 only (instantiation of the pattern).

**Pattern catalog browser**  The security patterns catalog is only available through the tool, and could not be printed. For the group without annotations, the browser to navigate the catalog is a simple HTML viewer with an alphabetically sorted index of all available patterns.

For the group with the annotated catalog, the browser additionally allows to filter the index according to a specific security objective, the applicability of the pattern, the trade-off labels of interest, or a combination of the above. Also, the annotations and relationships between the patterns are displayed in the pattern description.

**Measurement collection**  The tool is instrumented to monitor the team’s actions in the background. In particular, the tool collects the following information for each task:

- the time spent in each step of the process (study, shortlist, select and instantiate);
- the patterns that are browsed in the catalog;
- the patterns that are shortlisted and finally selected;
- the total time each pattern is looked at;
- the total time each editor (task viewer, pattern catalog, UML editor) is active;
- the modifications that are performed on the UML model of the system.

The tool is also equipped with a pause button, which the teams were instructed to use when they were taking a break. Pressing this button stops the time measurements and at the same time hides all information that is visible in the tool, until work is resumed.

The collected measurements are stored in an encrypted log file, and are automatically transmitted to a web server after the completion of every task. The teams were made aware of this beforehand, so they knew they had to be connected to the Internet when completing a task. However, they did not know precisely which measures were collected.
3.4.3 Towards a general experimental platform

While the tool was developed specifically for the experiment described above, it is feasible to evolve it into a platform to execute empirical experiments related to software engineering.

First, the configuration of the tool with respect to the active team, their treatment group and task sequence are easily reusable for different experiments. Also, the presentation of the tasks does not depend on the particular experiment, and can easily be re-used for other tasks. Furthermore, the measurement subsystem in the tool is written in a generic way, and allows to define additional types of measurement that can be collected in the same (encrypted) log file.

The process that the subjects have to follow is most likely to be specific for each experiment. If it is possible to describe the process using a generic model, for example as a state machine or an activity diagram, support can be offered to more easily enforce any process described in such manner. Nevertheless, substantial custom code will always be required for the process aspect of an experiment.

The tool currently does not have support for participants that work in group and collaborate using different instances of the tool, but such support may be desired for other experiments.

As a complement to the experiment execution tool, a tool for the experiment designer is also required. The purpose of this tool is to generate the necessary configuration files and other input for executing the experiment. For example, it can be used to distribute the subjects across different groups, randomizing the order of the tasks per subject, and persist these in an (encrypted) configuration file. Also, utilities should be available to process the collected data (aggregating, filtering, exporting, etc.).
4 Pattern-based Cloud Analysis

<table>
<thead>
<tr>
<th>Purpose of the prototype</th>
<th>ISO 27001 context description for cloud computing systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modeling formalism</td>
<td>UML-based</td>
</tr>
<tr>
<td>Security concern</td>
<td>Will be used as basis for ISO 27001 policy definition in the future</td>
</tr>
<tr>
<td>Foreground / background</td>
<td>The pattern is background and the tool is foreground.</td>
</tr>
</tbody>
</table>

Table 4.1: Synthetic positioning of a Cloud Pattern Analysis Tool

We explain the content of Table 4.1 and give a short overview of our contribution in this abstract. We explain the contribution in more detail in the following sections.

The purpose of our prototype is to ease the burden of producing the context description of the ISO 27001 [27] standard for cloud computing systems. Furthermore, a specified context establishment is the foundation for the realizing the further steps in the standard, e.g., realizing security policies. Thus, we are able to use this foundation for consistency checks. For example, we envision that a security policy must not contain any element that is not present in the context establishment. The context establishment is also fundamental for realizing traceability functions. For example, we want to design a function that lists all the security policies referring to a stakeholder of the system and the controls that are implemented to realize the policies. Hence, we can trace the security effort that is cause by this particular stakeholder. This information can be used for security reasoning, e.g., which stakeholder causes which security measures.

The context description in the ISO 27001 is an important first step in the standards implementation, because all the following steps use the information in the context description, e.g., the policy description, risk assessment, and selection of controls. We created a pattern that is based on UML and that can be instantiated for any given cloud computing system. The information in the context description will be used in future work for creating ISO 27001 compliant documents for e.g. policy descriptions. The foreground is the actual tool implementation and the background is the pattern, which we published before NESSoS.

4.1 Background about the ISO 27001 Standard

The ISO 27000 is a well-established series of information security standards, relevant for any kind of organization. The scope for applying these standards can be an organization as a whole, single business processes or even an IT application or IT infrastructure. The context establishment and the asset identification are among the first steps to be performed. The quality of the results produced when performing these steps has a crucial influence on the subsequent steps such as identifying loss, vulnerabilities, possible attacks or defining countermeasures. Thus, a structural contextual analysis to gather all necessary information in the initial steps is important, but is not offered in the standard. In this work, we focus on the scope of cloud computing systems and a way to support the context establishment and the asset identification described in ISO 27005 [28]. ISO 27005 refines the risk management part of the ISO 27001 standard.

The normative standard of the series, the ISO 27001, contains the requirements for an Information Security Management System (ISMS) [29]. The standard prescribes a process, which tailors security to the needs of any kind of organization. The remaining standards of the ISO 27000 series describe parts, or usage scenarios, of the ISMS in detail. For example, ISO 27005 [28] describes information security risk management. The ISO 27005 has a certain significance as the ISO 27001 is risk-centered in many sections and the ISO 27005 describes the risk assessment process and the risk documentation and management in detail. However, the ISO 27005 is not normative.

The Table 4.2 lists the content of Section 4 of the ISO 27001 standard, which states the requirements for an ISMS. The Cloud Pattern Analysis tool so far supports the Sect. 4.2.1 a, the description of the scope and boundaries of the ISMS, which includes the identification of assets. We are planning to support Sect. 4.2.1 b in the future, the establishment of an ISMS policy. The tool support will enable us to conduct consistency checks between the policies and the context description. For example, a stakeholder that is part of a policy has to be also part of the context description.

1Unified Modeling Language: http://www.omg.org/spec/UML/2.3/
Figure 4.1: Cloud System Analysis Pattern

Table 4.2: ISO 27001 [27] Section 4

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sect. 4.1</td>
<td>General requirements</td>
</tr>
<tr>
<td>Sect. 4.2</td>
<td>Establish and manage the ISMS</td>
</tr>
<tr>
<td>Sect. 4.2.1</td>
<td>Establish the ISMS</td>
</tr>
<tr>
<td>Sect. 4.2.1 a</td>
<td>Define scope and boundaries</td>
</tr>
<tr>
<td>Sect. 4.2.1 b</td>
<td>Define ISMS policy</td>
</tr>
<tr>
<td>Sect. 4.2.1 c</td>
<td>Define risk assessment</td>
</tr>
<tr>
<td>Sect. 4.2.1 d</td>
<td>Identify the risk</td>
</tr>
<tr>
<td>Sect. 4.2.1 e</td>
<td>Analyze and evaluate risk</td>
</tr>
<tr>
<td>Sect. 4.2.1 f</td>
<td>Identify risk treatment</td>
</tr>
<tr>
<td>Sect. 4.2.1 g</td>
<td>Select controls</td>
</tr>
<tr>
<td>Sect. 4.2.1 h,i</td>
<td>Obtain management approval</td>
</tr>
<tr>
<td>Sect. 4.2.1 j</td>
<td>Prepare a statement of applicability</td>
</tr>
<tr>
<td>Sect. 4.2.2</td>
<td>Implement and operate the ISMS</td>
</tr>
<tr>
<td>Sect. 4.2.3</td>
<td>Monitor and review the ISMS</td>
</tr>
<tr>
<td>Sect. 4.2.4</td>
<td>Maintain and improve the ISMS</td>
</tr>
<tr>
<td>Sect. 4.3</td>
<td>Documentation requirements</td>
</tr>
</tbody>
</table>
4.2 The UML4PF - Cloud Pattern Analysis Prototype

We published in [3] a cloud system analysis pattern and different kinds of stakeholder templates that serve to understand and describe a given cloud development problem, i.e. the envisaged IT systems and the relevant parts of the operational environment. The UML4PF - Cloud Pattern Analysis Prototype implements software support for these patterns and templates.

We present our support tool for the cloud system analysis pattern in the following. The pattern (see Fig. 4.1) provides a conceptual view on cloud computing systems and serves to systematically analyze stakeholders and requirements.

A Cloud is embedded into an environment consisting of two parts, namely the Direct System Environment and the Indirect System Environment. The Direct System Environment contains stakeholders and other systems that directly interact with the Cloud, i.e. they are connected by associations. Moreover, associations between stakeholders in the Direct and Indirect System Environment exist, but not between stakeholders in the Indirect System Environment and the cloud. Typically, the Indirect System Environment is a significant source for compliance and privacy requirements.

The Cloud Provider owns a Pool consisting of Resources, which are divided into Hardware and Software resources. The provider offers its resources as Services, i.e. IaaS, PaaS, or SaaS. The boxes Pool and Service in Fig. 4.1 cannot be instantiated, because these are just conceptual terms. Instead, the specialized cloud services such as IaaS, PaaS, and SaaS and specialized Resources are instantiated. The Cloud Developer represents a software developer assigned by the Cloud Customer. The developer prepares and maintains an IaaS or PaaS offer. The IaaS offer is a virtualized hardware, in some cases equipped with a basic operating system. The Cloud Developer deploys a set of software named Cloud Software Stack (e.g. web servers, applications, databases) into the IaaS in order to offer the functionality required to build a PaaS. In our pattern PaaS consists of an IaaS, a Cloud Software Stack and a Cloud Programming Interface (CPI), which we subsume as Software Product. The Cloud Developer prepares and creates SaaS offers based on the CPI, finally used by the End Customers. SaaS processes and stores Data in- and output from the End Customers. The Cloud Provider, Cloud Customer, Cloud Devel-
oper, and End Customer are part of the Direct System Environment. Hence, we categorize them as direct stakeholders. The Legislator and the Domain (and possibly other stakeholders) are part of the Indirect System Environment. Therefore, we categorize them as indirect stakeholders.

We accompany this cloud system analysis pattern by templates to systematically gather domain knowledge about the direct and indirect system environments based upon the stakeholders’ relations to the cloud and other stakeholders. These are described in detail in [3]. The template that serves to describe stakeholders contained in the direct system environment is shown in the following:

**Name** State the identifier of the stakeholder or group of stakeholders, e.g. company name or group of end customers.

**Description** Describe the stakeholder informally, e.g. if the stakeholder is a natural or a legal person.

**Relations to the cloud** Describe the inputs and outputs represented as relation (line from this stakeholder to the cloud) between the stakeholder and the cloud, e.g. the kind of data or software.

**Motivation** State the motivation of the stakeholder for using the cloud based on the previous considered relations to the cloud, e.g. business goals such as profit and costs reduction.

**Relations to other direct stakeholders** For each relation (line from this stakeholder to another direct stakeholder), name the kind of dependency between the stakeholders, e.g. controlled by contract, served by, indirectly influenced by customer-demand.

**Assets** Identify the assets relevant for this stakeholder, e.g. by considering the relations to the cloud.

**Compliance and Privacy** Identify relevant compliance and privacy laws as well as regulations based on the indirect stakeholders. Specify and identify the ones relevant for the stakeholder at hand, e.g., the German Federal Data Protection Act - BDSG.

The second template serves to describe stakeholders contained in the indirect system environment:
Name See direct stakeholder template.

Description See direct stakeholder template.

Relations to other stakeholders For each relation from this stakeholder to another direct or indirect stakeholder (no line explicitly shown), name the kind of dependency between the stakeholders, e.g. protected by, controlled by law, implement laws.

Motivation State the motivation of the stakeholder for having any reason of considering the cloud for its work or the motivation for having any kind of relation to stakeholders of the direct or indirect environment, e.g. protect privacy of citizens or implement concrete laws of an economic community.

Compliance and Privacy Identify relevant compliance and privacy laws and regulations for the cloud scenario, e.g. the German Federal Data Protection Act - BDSG and the Control and Transparency in Enterprises Act - KonTraG.

We provide modeling support as part of our support tool that allows to extend the cloud system analysis pattern with additional direct or indirect stakeholders, as well as further cloud elements, and the relations between them. Fig. 4.2 shows the additional stakeholder Cloud Phone Support.

We also added a validation functionality in the tool that checks, e.g., if newly added elements to the pattern have been given a name. In addition, the tool checks if the instantiation process has worked correctly. The tool also has a validation functionality for the instantiation of the pattern. For example, Fig. 4.3 shows several warnings, which state that the attribute compliance in several stakeholder templates is not entered.

The notation used to specify the pattern is based on UML\textsuperscript{2} notation, i.e. the stick figures represent roles, the boxes represent concepts or entities of the real world, the named lines represent relations (associations) equipped with cardinalities, the unfilled diamond represents a "part-of" relation, and the unfilled triangles represent inheritance.

The prototype offers pattern-based support for context establishment and asset identification for the ISO 2700x standards in the field of cloud computing. This approach serves as a proof-of-concept that security standards can largely benefit from patterns. Our approach comprises the following main benefits:

- Systematic pattern-based identification of assets and context establishment for clouds,
- Building a foundation for consistency checks for security artifacts in clouds, e.g., security policies,
- Establishing a foundation for traceability checks for security functions in clouds, e.g., which stakeholder causes the realization of which security control,
- Improve the outcome of a ISO 2700x implementations by perfecting the initial steps context establishment and asset identification.

Furthermore, we are planning to use the cloud analysis pattern as context elicitation stage for security requirements engineering approaches. In WP 6 we work with a pattern for Service Oriented Architectures (SOA), which we designed in a similar manner as the cloud analysis pattern. The information collected in the SOA pattern will serve as input for generating SI* models for requirements conflict analysis. Thus, we provide methods that include information about future Internet applications, e.g., clouds and SOA into the requirements analysis phase of software development processes.

We published the idea of the Cloud Analysis Pattern [3] before the NESSoS project. Hence, this paper is background. However, the development of the prototype for the pattern is done as an effort in NESSoS and constitutes foreground. We are planning to integrate the prototype into the SDE platform, once the implementation and testing is finished.

\textsuperscript{2}Unified Modeling Language: \url{http://www.omg.org/spec/UML/2.3/}

4.3 Implementation Overview

Basis for our tool is the Eclipse platform [15] together with its plug-ins Eclipse Modeling Framework (EMF) [17] and the Graphical Editing Framework (GEF) [18]. We use further the Graphical Modeling Framework (GMF) [16], which provides a set of generative components for developing a graphical editor based upon EMF and GMF.

We present our architecture in Fig. 4.4. The Cloud Pattern Analysis Eclipse Plugin uses the GMF framework for creating the graphical user interface for the Cloud Pattern Analysis. We created a EMF model of the Cloud Pattern that we will explain in the following. The model itself is stored in the CAP module. CAP stands for Cloud Analysis Pattern and stores the diagram parts of the tool. This part also stores models that are created using the graphical user interface (GUI).

This model is used by the GMF framework, which uses in turn the combination of the EMF and the GEF framework to generate the source code for the cloud pattern analysis GUI. The UML4PF - Cloud Pattern Analysis Tool provides the functionalities to create cloud patterns and to instantiate existing cloud patterns. The Cloud Pattern Analysis Plugin uses the eclipse interface IWizard to create a wizard to support the instantiation of an existing cloud pattern. The wizard provides a graphical interface that asks the user for the information necessary to instantiate a cloud element or a stakeholder, e.g., the name of a stakeholder. In addition, the wizard supports the instantiation of several instances of a cloud element or a stakeholder. For example, the wizard can instantiate five cloud customers at once. Furthermore, the wizard can validate an instantiated cloud model. This validation consists at the moment of a check if all template fields of stakeholders have entries.

The tool can also export instantiated models to a pdf file. This file is called report and contains the graphical model of the instantiated pattern as well as the text in the stakeholder templates. We use the iTextPDF interface of Eclipse for the PDF creation.

We present a UML class model of our Cloud Analysis pattern in Fig. 4.5 that is created using EMF. The initial element is the class for the cloud analysis pattern (CAP) diagram. Each CAP diagram has at most one Indirect Environment, which can have Indirect Stakeholders. The indirect stakeholders inherit from the class Stakeholder. Stakeholder requires attributes for the field of the stakeholder templates. Indirect
and Direct Stakeholders share the attributes name, description, motivation and compliance and privacy in their templates. The stakeholder class contains also an attribute for quantity. This states the amount of stakeholders of a kind in the model and is not used in the template. This field is used by the Wizard to support the instantiation process.

An indirect environment has at most one direct environment, which contains direct stakeholders and the Cloud. The cloud contains Cloud Elements that have a name. Direct Stakeholders have relations to cloud elements and vice versa.

The graphical editor, which is generated using this EMF model, has several constraints on the modeling GUI. These are automatically derived due to the structure of the model. For example, the first element that a user can create has to be an indirect environment. Furthermore, indirect stakeholder can not have relations to direct stakeholders or cloud elements. In addition, indirect stakeholders can only be placed in the indirect environment.
5 Combining Change Patterns and Runtime Adaptation

<table>
<thead>
<tr>
<th>Purpose of the prototype</th>
<th>Co-evolve access control policies, architecture and a running system</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modeling formalism</td>
<td>Ecore (Problem Frames, Kevoree (as ADL))</td>
</tr>
<tr>
<td>Security concern</td>
<td>Access control</td>
</tr>
<tr>
<td>Foreground / background</td>
<td>New, extended implementation of an earlier prototype</td>
</tr>
</tbody>
</table>

Table 5.1: Synthetic positioning of change patterns for runtime adaptation prototype

Over time, the access control policies that need to be enforced by a software system may change. For example, new attributes may be introduced that need to be taken into account when making access decisions. These changes impact the system, and thus need to be propagated, though the architecture, up to the actual running system. The developed prototype shows how change patterns and runtime adaptation can be combined and used to support such evolution of access control policies.

5.1 Background

Once software is deployed into a production environment, it may remain in place for a long time, often longer than originally anticipated by the development team. Over time, though, the context in which that system is used may change. That context is comprised of multiple factors, such as the primary goal of the system, the set of people that have access to the system, or the legislation to which the system must comply. A new context likely necessitates changes to the software, like adapting its behavior in certain situations, extending its functionality by introducing new components, modifying the connections between the components, or a combination thereof. These changes have to be propagated from the point where they are made (e.g., the requirements), through the intermediary artifacts (e.g., architecture, design, code, etc.), up until the running system that eventually has to incorporate them.

5.2 Problem statement

For this prototype, we focus on access control policies as the changing context. So far, there is no support for propagating a change in the set of available entities in the policy, though the architecture, up to the actual running system. Examples of such change are the introduction of a new attribute for a set of subjects, or the addition of a new resource that needs to be protected. These changes can be considered as changes to the requirements of the system, because the system must provide the necessary functionality that enables the enforcement of access control policies that refer to these entities. Our prototype shows how such propagation from requirements to runtime can be achieved.

5.3 Approach

The prototype is based on the concept of change patterns [47], an adaptive middleware platform named Kevoree [21] \(^1\), and the modeling of access control policies in the Problem Frames formalism [30].

A generic kind of change at the requirements level is captured in a change pattern by means of a change scenario, which consists of a pair of requirements templates that describe, in a generic way, the situations before and after the anticipated change. Moreover, the pattern provides a collection of architecture-level solutions that enable the system to respond to the change, while minimizing the effort required to evolve the architecture. The solution contains a generic architectural template, and a transformation based on that template, called the guidance.

\(^1\) more information available on: http://kevoree.org/
In our prototype, the access control policy (modeled using Problem Frames) is the artifact in which a change scenario occurs. The changes are propagated to the system's runtime architecture by executing the corresponding guidance transformation on a Kevoree model.

5.4 Prototype

The developed prototype shows how change patterns and runtime adaptation can be combined and used to support the evolution of access control policies. To illustrate the use of the prototype, we will use a running example from the health care domain. This example is first presented in the next subsection. Then, the prototype is described from the viewpoint of its two main stakeholders. The last subsection gives more detail on how the prototype is structured.

An earlier, similar prototype implementation concerning change patterns already supported co-evolving a trust model (expressed in Si*) and an architectural model (expressed in UML). The prototype presented here is inspired by that earlier prototype, but is a new implementation. Additionally, it provides a framework for the pattern creator, making it easy to define new patterns. Furthermore, it contains usability improvements for the end user (the modeler), and it is more flexible with respect to the kind of models and matching techniques that can be used. The prototype also demonstrates an automated end-to-end solution to deal with change, by adapting the running system based on changes made in the requirements model.

5.4.1 Running example

As a running example, we use the development of electronic health care system. For presentation purposes, we limit ourselves to a very simple system, namely one that allows a doctor to view the electronic health record (EHR) of a patient. This system is to be extended with support for emergency situations by means of ‘emergency levels’ [7]. When a certain emergency level is made active, additional policies become enabled that grant more permissions to the subjects that are using the system. In practice, the system needs to be extended with functionality to set the current emergency level, and should make the current emergency level available as an (environment) attribute to the PDP.

The Problem Frames model of the example system (before the notion of an emergency level is introduced) is shown in Figure 5.1(a). It shows a system that consists of an EHR Viewer (the machine domain), which a doctor can use (via a terminal) to view a patient’s EHR. This access is regulated by means of a policy, which currently can refer to the doctor’s and patient’s identity attribute only.

A simplified model of the Emergency Level subsystem is shown in Figure 5.1(b). It consists of a management machine domain that can retrieve and update the current emergency level. To connect both subproblems, the current emergency level should be made available to the policy, as shown in the right hand part of Figure 5.1(c).

The initial architecture of the example, as displayed by the Kevoree editor, is shown at the top of Figure 5.2. It has three nodes, one for the client, one for the business logic and one where the data is stored. On these nodes, several components are deployed, such as the doctor’s client GUI, the eHealth system, the policy decision point, and a user database. Also, one of the nodes needs to contain a component dedicated to the change patterns approach, as discussed in Section 5.4.3. The bottom of the figure shows the final architecture, i.e., after the emergency level attribute is made available to the policy decision point. This architecture was automatically obtained by following the change patterns process triggered by modifying the access control model.

5.4.2 Stakeholders and supported tasks

The prototype supports two main stakeholders. The first (and primary) stakeholder is the end user of the prototype. In the context of the tool, this is the person that models the requirements of the system. In the e-health system, for example, this person needs to add the concept of emergency level to the Problem Frames model, specify how the emergency level is managed, and specify that the current emergency level can influence access control decisions.
The second stakeholder supported by the prototype is the creator of change patterns. He does not directly interact with the tool, but uses a framework (i.e., an API) that allows him to define patterns that can be used by the modelers. For instance, he should define the pattern (consisting of a scenario and a solution) for dealing with a new environment attribute such as the emergency level.

The rest of this subsection demonstrates how the prototype supports the tasks of these two stakeholders.

**Modeler**

When the modeler modifies the requirements model of the system, other models of the system (e.g., the architecture) should be updated accordingly, so that all models remain consistent. Eventually, the running system should be updated as well so that the model describes the actual system.

The developed prototype supports the modeler in maintaining this consistency over two models, in particular a Problem Frames model for modeling access control, and an architectural model. It does this by continuously monitoring the model that is being edited, based on the available set of change patterns. In the e-health example, the modeler edits the original Problem Frames model (Figure 5.1(a)) in Eclipse, and augments it with the concept of an emergency level (Figure 5.1(c)).

Adding the emergency level corresponds with the initial situation described by the change pattern dealing with a new environment attribute. This fact is automatically detected and brought to the attention of the modeler, who can then choose to either select or ignore that particular instance. To this aim, the prototype contains a change pattern dashboard view. Figure 5.3 contains a screenshot of this dashboard. When a new instance of a pattern has been found, this is indicated by adding the instance in the leftmost list ("Undecided instances").

The modeler can then include the newly detected pattern instance for further analysis, or choose to ignore it. In practice, selecting an instance for further analysis means that the model is kept being monitored to detect the occurrence of the anticipated situation for that instance. To judge the relevance
Figure 5.2: Initial (top) and final (bottom) architecture of the eHealth example in Kevoree
of the pattern instance, it can be opened for inspection, resulting in the editor shown in Figure 5.4. This editor shows the name and a description of the instance, and shows the elements from the model that play the roles of the template defined by the change pattern scenario.

At the same time, if the pattern is not ignored, one or more solution instances can be attached. A solution instance corresponds to a transformation of another model, for example the architecture of the system. In Figure 5.4, it is visible that there is one available solution, specific to the Kevoree platform, namely connecting the policy decision component with the attribute provider component.

If the modeler picks this solution, he has to link the generic solution template to the actual architecture of the system, by specifying a binding between the roles of the template and the elements in the system architecture. Concretely, the modeler has to select, for each role of the template, the Kevoree model element of the running system that should play this role, as shown in Figure 5.5. The solution is now prepared so that it is ready to be applied in the future (if necessary).

When the modeler subsequently edits the access control policy model, and adds a reference from the policy to the current emergency level (i.e., updating the model to the one shown in Figure 5.1(c)), this is detected as an occurrence of the previously selected change pattern instance. The modeler is then prompted to apply the prepared solution. When the solution in the eHealth case is applied, the Kevoree model is adapted by adding an instance of the Emergency Manager component, and adding a binding between the PDP component and this new component. This new model is deployed, and from that point on, the emergency level attribute is available to the policy evaluation engine. Note that, in the implemented example, the actual running system is updated automatically. While it may not always be necessary (or even desirable) to do this, it demonstrates the powerful usage scenarios that become available by using the techniques implemented in the prototype.

**Pattern creator**

The patterns that were used by the modeler had to be defined beforehand by a pattern creator. This stakeholder can use the framework of the tool to define patterns in an easy manner. Defining a pattern consists of two parts: defining the change scenario, and defining one or more solutions for that scenario. Scenarios and solutions are coupled using the extension mechanism of Eclipse, so it is also possible to define only a scenario, or to define a solution for an already existing scenario. In this way, existing change pattern catalogs can be extended with new solutions simply by installing a new plugin.

To define a change scenario, the pattern creator starts by creating the two requirements templates of the change scenario, namely the situation before and after the change. These templates can be specified in any suitable manner, as long as there is a method to find all matches of a template with an actual model. The prototype offers support for specifying patterns using EMF IncQuery [4], but other techniques can easily be added.

To define a solution, the pattern creator again creates two templates, namely of the situation before and after the solution is applied. Additionally, a transformation must be defined that converts a model that
Figure 5.4: Change pattern instance details

Figure 5.5: Preparing a solution by specifying the binding
matches the first template to a model that matches the second template. Again, transformations can be specified using any available technology. For the running example, a solution is defined that connects a PDP component with an Attribute Provider component by adding a binding between their ports. The transformation is performed by generating and executing a Kevoree reconfiguration script, which updates the architecture of the running system.

To make the pattern available to other users, two factory classes should be implemented, based on classes from the change pattern framework. One factory is responsible for creating the scenario, the other for the solution. These factory classes are placed in an Eclipse plugin, and registered as an extension to the change pattern framework using the Eclipse extension mechanism. Other users can now use the pattern simply by installing the plugin.

5.4.3 Under the hood

The architecture of the prototype is shown in Figure 5.6.

The left hand side shows the Eclipse environment, which is used for modeling. This environment contains the plugin for the change pattern framework (displayed in green). Additionally, it contains the plugin that defines change scenarios for access control modeled in problem frames, and the plugin that defines solutions (based on Kevoree) for these change scenarios.

The right hand side shows the Kevoree environment. It consists of several nodes, which are connected through the Kevoree middleware framework. One of the nodes contains a component that makes the Kevoree model of the running system available via CDO. This link enables the modeler to refer to Kevoree elements in Eclipse when specifying the solutions, and allows applying the transformations defined by the solutions to the running system.

The automated detection of change pattern instances is based on pattern matching, i.e., detecting bindings between the before and after templates of a scenario and the model. In the prototype, the pattern matching is performed by EMF IncQuery, which enables efficient pattern matching on EMF models. Patterns are specified in a dedicated language, from which code is generated that can be used to retrieve the matches for a model. The matching is incremental, i.e., a change to the EMF model does not require all patterns to be matched again. Instead, only the matches that can possibly have been influenced by the applied change are recomputed. In this way, pattern matching remains nearly instantaneous, even for very large models.

\(^2\text{CDO is an Eclipse technology to enable distributed, concurrent access to a single EMF Ecore model in a repository. More information is available on http://www.eclipse.org/cdo.}\)
6 Access control on pervasive architectures

<table>
<thead>
<tr>
<th>Purpose of the prototype</th>
<th>Access-control on pervasive architectures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modeling formalism</td>
<td>RBAC and KEVOREE</td>
</tr>
<tr>
<td>Security concern</td>
<td>Access-control (authorization)</td>
</tr>
<tr>
<td>Foreground / background</td>
<td>KEVOREE, RBAC and models@runtime are background, mapping and prototype is foreground</td>
</tr>
</tbody>
</table>

Table 6.1: Synthetic positioning of the approach

Pervasive systems typically involve heterogeneous users, devices and networks to permit users seamlessly interacting between them and with the physical world. In order to be flexible, these systems must be both dynamically adaptive to handle changes occurring at runtime and still open to the ability of receiving new elements. Characteristics of these systems can have a major impact on the enforcement of role-based access control policies (RBAC). Enforcement mechanism for RBAC policies need to be tailored to distributed and adaptive software architectures. The mechanism must be capable of handling architectural changes (e.g. a resource hosted by a node is moved to another node) in order to maintain the enforced policy. In this chapter we describe an approach of policy enforcement that leverages on a mapping between RBAC and a component-based architecture (kevoree) to reason on architectural changes and maintain the enforced policy. The Models@runtime paradigm provides elementary bricks to reason about adaptive architecture. Relying on this paradigm and on runtime adaptation and monitoring mechanisms we present a prototype for an RBAC enforcement mechanism.

6.1 Background

6.1.1 Models@runtime paradigm

Models@runtime paradigm [5] proposes to keep a running system synchronized with an abstract model representing it. Based on this principle, some approaches have described ways to support the dynamic adaptation of a system using the reflective model [38]. Working on an abstract model representing a running system allows applying various adaptation strategies in an off-line mode in order to generate a target model. It also permits to validate the target model in an on-line mode, before the adaptation of the system. The target model can finally be used to automatically generate the adaptation script allowing to modify the running system such that the new system configuration corresponds to the target model.

6.1.2 RBAC (Role-based access-control)

RBAC is a NIST standard [19] that describes an access-control model allowing to specify a policy around the concept of role. Users are assigned to roles and roles are associated with permissions. Permission allows specifying a set of operations which can be applied on a set of objects. A session contains references of roles that are activated by users. A user is authorized to perform an operation on an object if its session contains the role associated with a permission allowing the operation on the object. An important feature of RBAC is the separation of duty (SoD), allowing to state that two or more roles can not be assigned to a same user (SSoD : Static SoD) or that two or more roles can not be activated by a same user (DSoD : Dynamic SoD). SoD enforces that more than one user is required to perform actions that may have critical consequences [42].

6.1.3 Kevoree

Kevoree [21] is a component model for designing dynamic distributed systems. This component model is embedded in a complete tool-suite for modeling and deploying adaptive systems. Six main concepts constitute the basis of the Kevoree component metamodel (Node, Group, Component, Port, Binding and Channel). A node represents a device on which software components are deployed. A group allows defining a set of nodes which are sharing the same representation of the reflecting architectural model. A
port represents an operation that a component provides or requires. A binding represents communication channel between a port and a channel. A channel is also a concept in the Kevoree component metamodel and represents a semantic of communication. The Kevoree core library proposes an implementation of these concepts for various kinds of platforms (Java, Android, Arduino, ...) [20]. The adaptation of the system can be performed using a set of primitives allowing to describe how to add and remove architectural elements in the model.

6.2 Problem statement

The management of access-control concern is complex in pervasive systems, the policy enforcement mechanism has to be tailored to deal with system changes. Example of such changes are the addition/removal of nodes, resources or services. Dynamism and heterogeneity (e.g. a node can be a laptop, smartphone, tablet, sensor, ...) characteristics of these systems imply that the access-control mechanism must be able to cope with different configurations which may occur in an unforeseeable way. The unexpectedness of these changes precludes an a priori specification of how the mechanism has to deal with a system configuration. The problem is the lack of techniques permitting to reason about and handle these changes.

6.3 Approach

The proposed approach consists of an access-control mechanism that can monitor architectural changes and automatically handle them in order to maintain the enforced policy. From one side a security expert is in charge of designing the RBAC policy with additional attributes permitting to integrate some architectural information available from the system design. From the other side the software architect models the distributed adaptive system using Kevoree. The enforcement strategy consists in the addition/deletion of connections between users and resources according to allowed actions for each activated policy rules, the user can thus apply all activated policy rules. The main difference with the enforcement strategy of XACML implementations is that the enforcement is not based on a request evaluation each time a user wants to apply a rule.

6.3.1 Access-control mechanism

The access-control mechanism consists of a component instantiated on a node of the system. It is in charge of monitoring whether the architectural model enforces the system policy and to handle actions permitting to maintain the enforced policy. Figure 6.1 illustrates the RBAC enforcement approach, the mechanism is represented in the green rounded box on the right. It is represented on the model layer because it only interacts with the reflecting model of the system. The mechanism is tied to the architectural monitoring mechanism in order to be notified when the system changes and to have an access to the reflecting architectural model (RAM) of the system. The enforcement approach consists in generating a target architectural model (TAM) such that it enforces correctly the policy. The access-control mechanism is also tied to the runtime adaptation mechanism for managing the adaptation of the system according to the TAM that is generated. To summarize, the access-control mechanism consists of a component having the ability to analyze and adapt an architectural model according to the policy enforcement concern by following a set of rules described in a mapping that we present in the next part.

6.3.2 Mapping between RBAC and Kevoree

In this part we describe the relations between RBAC and Kevoree concepts. These relations are used by the access-control mechanism to manage the adaptation of the architectural model such as it enforces the policy.
RBAC user

Each active user is associated with a node. Each user device (node) can be used only by the user owning it. A user instance of the policy has a corresponding node instance in the architecture if the user is connected to the system. The architectural change corresponding to the connection/disconnection of a user is the addition/removal of a node into the RAM.

RBAC role

A role is associated with a component type. This component type represents the application that the user can use when he activates a role. Each node representing a user can contain one instance of each role that the user can activate simultaneously. The activation of a role requires verification with respect to the DSoD property.

RBAC operation

An operation is associated with component's ports. In the case the component type represent a role, operation is a provided port and in the case the component type represent an RBAC Object, the operation is a required port.

RBAC object

An object is associated with a node and a component type, the node is the device hosting it and the component type represents the kind of object that can be instantiated. Two level of granularity can be defined for the object, it can represent either a type or an instance. In the case that the object represents a type, operations associated with it are authorized for each instance of this type. In the case the object
represents an instance, operations associated with it, can only be performed on this instance. The architectural change corresponding to the addition/removal of an object is the addition/removal of a component instance.

RBAC rule

An RBAC rule is the quadruplet [user, role, operation, object]. The rule is associated with one channel and two bindings. The first binding is between the role component instance port representing the operation from the user node and the channel. The second binding is between the object component instance port representing the operation from the object node and the channel. For the sake of simplicity, these two bindings and the channel are called a connection. The architectural change corresponding to the addition/removal of a rule is the addition/removal of a binding.

RBAC session

The session is not associated with an architectural element but can be inferred by analyzing components instantiated in a user node. If the component type of the instance corresponds to a role then this role is considered activated. The user is in charge to activate or deactivate roles. By default, when a user connects to the system a session empty of roles is added in the policy. The architectural change corresponding to the activation/deactivation of a role is the addition/removal of a component instance in a user node.

6.3.3 Access-control reasoner

The access-control reasoner is the part of the mechanism in charge to produce the adaptation script in order to maintain the enforced policy. It relies on two monitors permitting to observe architecture and policy models. Each monitor permits to detect predefined type of changes which may have an impact on the policy enforcement. Detection of each type of change is associated to actions permitting to handle it according to the policy enforcement concern.

Let's consider for example the instantiation of a component into a node. When an occurrence of such change is detected, its signature (e.g. the name and the type of the node instance, the name and the type of the component instance) permits to determine whether it concerns a user-role activation, in the addition of a resource or has no impact on the policy enforcement. In this example the change consists of a component instantiation and corresponds to a role activation, first it will be checked that the role activation is allowed according to the DSOD property. In case the activation is allowed, the role is added to the session of the user having activated it. The policy monitor detects the role activation, the change is associated to actions permitting to enforce rules associated to this role into the architecture. The enforcement consist in the addition of connections in the architecture model permitting to let the user perform all operations on resources corresponding to rules associated to the role newly activated.

In this example, both monitors are used, the architecture monitor detect first a change where the associated action consists in modifying the policy with the addition of role reference to a session. Then the policy monitor detect the change and the associated action consists in modifying the architecture. Also in order to grant the role activation, the DSoD verification is required, according to the verification result various behaviors could be adopted. Additional details of the approach can be found in [12].

6.4 Prototype

We realized the prototype using the Eclipse Modeling Framework (EMF) \(^1\) for the design of the RBAC metamodel and for generating the code permitting to edit policy models. The Kevoree metamodel can be downloaded as an Ecore model (e.g., an Ecore model represents a class diagram conform to the Ecore metamodel) directly from the Kevoree project repository \(^2\).

\(^1\)http://www.eclipse.org/modeling/
\(^2\)https://github.com/dukeboard/kevoree/tree/master/kevoree-core/org.kevoree.model/metamodel
For the access-control reasoner, monitors have been realized using IncQuery [4] (e.g., IncQuery is a tool permitting to specify and perform incremental queries over EMF models). A monitor is in charge to detect architectural changes and the other one is in charge to detect policy changes.

Prototype is still under development and currently consists in a set of patterns permitting to manage user connections, roles activations and objects additions. It requires yet to be tested and evaluated.


7 ActionUWE: Transforming UWE to ActionGUI

<table>
<thead>
<tr>
<th>Purpose of the prototype</th>
<th>Transformation from UWE to ActionGUI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modeling formalism</td>
<td>UWE Models</td>
</tr>
<tr>
<td>Security concern</td>
<td>RBAC and navigational access control</td>
</tr>
<tr>
<td>Foreground / background</td>
<td>We started to develop a prototype for the transformation</td>
</tr>
</tbody>
</table>

Table 7.1: Synthetic positioning of ActionUWE

Both, UWE (UML-based Web Engineering) and ActionGUI are web engineering approaches for modeling secure web applications. They provide a graphical notation for the representation of the models: a UML profile for UWE and a proprietary notation for ActionGUI. UWE focuses on a high level of abstraction, whereas ActionGUI models can directly be transformed to code. In this report, we describe ActionUWE: the model-to-model transformation from UWE into ActionGUI models. As a result, our new ActionUWE approach allows modelers and developers to generate semi-automatically secure code from high-level UWE models.

7.1 Background

To secure web applications is increasingly important because of rising cybercrime as well as the growing awareness of data privacy. Besides confidential connections and authentication, both data access control and navigational access control are the most relevant security features in this field. However, adding such security features to already implemented web applications is an error-prone task. Therefore, the goal is to include security features in early stages of the development process, i.e., at requirements specification and modeling level.

Existing approaches, such as OOHRIA [36], OOWS [45], WebML [37], UWE [33, 9], or ActionGUI [2] already provide well-known methods and tools for the design and development of web applications. Most of them follow the principle of “separation of concerns” using separate models for different views on the application, such as e.g. content, navigation, presentation and business processes. Most of the available methods do not support the modeling of security, whereas the UWE approach by Koch et al. [9] and the ActionGUI approach by Basin et al. [2] define models for security features like access control. ActionGUI’s proprietary notation comprises data models, security models and GUI models, and the application logic is represented using OCL. UWE provides a set of UML stereotypes for each view defining a so-called UML profile. UWE’s main focus is on the process of discussing and planning an application from different points of view as e.g. requirements, content (data model), navigation, users and roles, basic rights, presentation and process.

7.2 Problem Statement

At the moment, no model-driven solution for secure web applications exists which can unite the advantages of both approaches, i.e.:

- the advantages of the high-level of abstraction of UWE with its many views (separation of concerns), helping the modelers and developers to plan and implement the application without referring to concrete technologies

- the advantages of a concrete modeling language like ActionGUI which is based on a formal specification of the whole application logic and its access control policies. Furthermore, those policies allow the generation of secure web applications where the security policies are automatically embedded in the GUI.

Our model-driven approach, called ActionUWE (using UWE and ActionGUI together) combines both approaches, enabling web engineers to model security issues for web applications in the abstract way of...
UWE and to transform this representation to ActionGUI. The ActionGUI model has to be enriched with a rather small amount of additional information to specify the details of the application's behavior. Afterwards validation checks that are available for ActionGUI models can be used to examine the model before it is subjected to the model-to-code transformation. Consequently, ActionUWE is the first approach to semi-automatically generate secure software from UWE models. More detailed information about ActionUWE can be found in [8].

At the moment, we are working on the implementation of the ActionUWE transformation. This prototype, called ActionUWEtransformer, will serve as a proof-of-concept.

7.3 The ActionUWE Approach

In this section, we briefly describe UWE and ActionGUI, before giving an overview of the ActionUWE transformation which transform a UWE model to an ActionGUI Model.

7.3.1 UWE and ActionGUI Models

As already mentioned in the introduction, ActionUWE aims at comprising the advantages of both, UWE and ActionGUI. UWE is based on the UML standard, i.e., UWE models can be edited with all UML editors that support UML profiles. UWE provides many different models that describe the web application from several abstract points of view. The focus is not on modeling every detail of the application, but on providing an overview of several aspects.

By contrast, the approach of ActionGUI is to model a web application in detail (with a proprietary emf/gmf eclipse plugin) so that a concrete web application can be generated afterwards. For this aim, ActionGUI uses a SecureUML+ComponentUML Model to specify access control rules and defines the whole web application by modeling the GUI enriched with OCL statements defining the application logic.

For the ActionUWE transformation, the following UWE models are useful (cf. left part of Figure 7.1):

The Requirements Model defines requirements for a project and can be used to get a better understanding. However, it is not necessary for ActionUWE itself.

The UWE Content Model contains the data structure, i.e. the data that is used by the application.

The UWE Role Model defines a hierarchy of user groups to be used for authorization and access control issues. It is usually included in a User Model, which specifies basic structures, e.g., that a user can take on certain roles simultaneously.

The UWE Basic Rights Model describes the security policy using role based access control. It constrains elements from the UWE Content Model and from the User Model.

The UWE Presentation Model models graphical parts of the web application.

The UWE Navigational States Model depicts the navigation flow of the application and navigation-related access control policies. Additionally, it comprises a Navigational Menu Model, including available menu entries of the application, regardless of their layout.

Further information about UWE models can be found at the UWE website1.

ActionGUI comprises the following models (cf. right part of Figure 7.1):

The ActionGUI Model contains not only the graphical layout of the application, but also the application logic, which is specified using OCL.

The ComponentUML Model describes the data structure.

The SecureUML Model defines a role based access control policy.

For more detailed information about ActionGUI, see [2].

---

1UWE website. [http://uwe.pst.tlf.ifi.lmu.de](http://uwe.pst.tlf.ifi.lmu.de)
7.3.2 ActionUWE

The ActionUWE model-to-model transformation relates the UWE model elements to those of ActionGUI in four steps:

**Step 1** maps the data structure and creates a main window for the web application and transforms menus modeled with UWE.

**Step 2** converts the remainder of the UWE Presentation Model into ActionGUI.

**Step 3** transforms the navigational flow information from the UWE Navigational States Model to ActionGUI Model.

**Step 4** maps security features.

As ActionGUI and UWE use a different way of grouping features to models, the ActionGUI model itself contains most of the transformed elements:

**The UWE Content Model** is mapped in a straightforward way to a ComponentUML Model in ActionGUI.

**The UWE Basic Rights Model** is mapped to a SecureUML Model in ActionGUI.

**The UWE Presentation Model** is mapped to a set of *widgets* that are part of an ActionGUI Model.

**The UWE Navigational States Model** is mapped to certain *Action* and *Event* elements of the ActionGUI Model.

7.4 Prototype

The *ActionUWE* transformer prototype is still work-in-progress, as the ActionUWE specification [8] was finished recently, as a collaboration between LMU and IMDEA in September 2012.

Our first goal is to further enhance ActionUWETransformer so that it can transform an address book example which we have modeled in UWE. The first version of this model does not contain menus, as the implementation of the menu transformation algorithm is rather complex and was postponed for a second step.

---

2 Address Book example: [http://uwe.pst.ifi.lmu.de/exampleAddressBookWithContentUpdates.html](http://uwe.pst.ifi.lmu.de/exampleAddressBookWithContentUpdates.html)
Technically, our implementation is based on the Eclipse M2M project\(^3\), which is a sub-project of the Eclipse Modeling Project.

\(^3\)Eclipse Model To Model (M2M). [http://www.eclipse.org/m2m/](http://www.eclipse.org/m2m/)
8 Verification of declarative ATL transformations

In Table 8.1, we summarize the main applications of our ATL2FOL prototype to security concerns as we illustrate below.

<table>
<thead>
<tr>
<th>Purpose of the prototype</th>
<th>Verification of declarative ATL transformations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modeling formalism</td>
<td>MOF compliant metamodels, OCL, ATL</td>
</tr>
<tr>
<td>Security concern</td>
<td>Preservation of security properties captured as relations between model elements or written as OCL pre- and postconditions through model transformations</td>
</tr>
<tr>
<td>Foreground / background</td>
<td>Half developed within NESSoS</td>
</tr>
</tbody>
</table>

Table 8.1: Synthetic positioning of the verification of declarative ATL transformations

8.1 Background

Model Driven Engineering (MDE) is a software development process where models constitute pivotal elements of the software to be built. If models are well-specified, transformations can be employed for various purposes, e.g., to produce final code. However, transformations are only meaningful when they are ‘correct’: they must produce valid models from valid input models. A valid model has conformance to its meta-model and fulfills its constraints, usually written in OCL. Otherwise, errors introduced by transformations will be propagated and may produce more errors in the subsequent MDE steps. Thus, well-founded and, at the same time, practical verification methods and tools are important to guarantee this correctness.

8.2 Problem Statement

A developer who is designing a model transformation typically wonders the following question several times during the designing process: Do the constraints imposed on the source model plus the transformation specification guarantee that the constraints imposed on the target model hold? When the answer to this question is ‘yes’ for certain properties, we would say that the transformation which is being designed is correct with respect to the given sets of pre and postconditions. Namely, in our view, a model transformation is correct if and only if executing it using a constrained-valid input model as argument always results on a constrained-valid output model, where a constrained-valid input model is a model that satisfies the model transformation’s preconditions and a constrained-valid output model is a model that satisfies the model transformation’s postconditions.

Our prototype is based on a novel methodology to perform automatic, unbounded verification of ATL [32] transformations. ATL (ATL Transformation Language) is a model transformation language and toolkit (http://www.eclipse.org/atl/). This is a research result of the collaboration between ATOS and INRIA during this second year of NESSoS. Our work focuses on checking partial correctness of declarative, rule-based transformations between constrained metamodels. More specifically, we consider the ATL transformation language and MOF [41] style metamodels that employ OCL constraints to precisely describe their domain. These ingredients are very popular due to their sophisticated tool support, and because OCL is employed in almost all OMG specifications.

8.3 Approach

The key components of our approach are a novel first order semantics for a declarative subset of ATL, based on the interpretation of ATL rules as first-order functions and predicates, and the use of automatic decision procedures for Satisfiability Modulo Theories problems in SMT solvers. Such solvers, e.g. Z3 [13, 49] and Yices [14, 46], have been significantly improved in the past years and can automatically and
efficiently decide several important fragments of first-order logic (FOL). Although, our semantics is not complete, it does cover a significant subset of the ATL language. Using this semantics, transformation correctness can be automatically verified with respect to non-trivial OCL pre- and postconditions by using SMT solvers. In particular, we regard the subset of OCL that is covered in [11]. Our aim is to provide a ‘push button’ technology that can be applied regularly in model transformation development by developers lacking of a formal background. Our approach combines the advantages of formal verification (in the sense that we aim to provide formal proofs) and automatic verification (in the sense that we do not require the transformation developer to operate, for example, interactive theorem provers).

8.4 Prototype

In the field of MDE, ATL provides ways to produce a set of target models from a set of source models. Developed on top of the Eclipse platform, the ATL Integrated Environnement (IDE) provides a number of standard development tools (syntax highlighting, debugger, etc.) which aim to ease development of ATL transformations. The architecture of our ATL2FOL prototype uses this toolkit as a front-end since it eases the use for the developer and because of the standard development tools that allow also the parsing and type checking of ATL transformations and OCL constraints. Thus, taking as input an ATL specification, i.e., a source and a target metamodel and an ATL transformation defined between them, our ATL2FOL prototype

(i) maps the OCL constrained metamodels to Z3, using the OCL2FOL component that implements the mapping defined in [11];

(ii) maps the ATL rules to Z3, following the novel first order semantics that we propose in [10], and

(iii) feeds and runs the Z3 SMT solver to check the unsatisfiability of the resulting specification.

As it is shown in Figure 8.1, the ATL2FOL prototype consists of 4 components: the ATL toolkit, the OCL2FOL component, the ATL2FOL component and the Z3 SMT solver.

In Figure 8.2, we show the interaction between the components and the activities that are undertaken by the user and by each component:

• ATL toolkit (http://www.eclipse.org/atl/) provides the developer with a user friendly interface to design OCL-constrained source and target metamodels and ATL transformations between them. This component parses and type checks the designed specification.
the OCL2FOL component blindly receives the OCL constrained metamodels and produces a file containing its corresponding FOL specification ready to feed Z3;

- the ATL2FOL component blindly receives the ATL transformation specification in XML format and extends the previous file with the corresponding FOL specification ready to feed Z3. This component generates one Z3 input file for each postcondition specified on the target metamodel, so the Z3 solver can perform all the required satisfiability checks.

- Z3 SMT solver receives the file and performs the unsatisfiability checks.

At the end of the process the user may manipulate the output file to use some of the postconditions as lemmas if Z3 did not succeed with some of the proofs.

**Example**

**Metamodels.** Figure 8.3 depicts the ER and REL metamodels that are (resp.) the source and target metamodels for the ER2REL transformation, which is depicted in Fig. 8.4. In the ER metamodel, a schema may have entities and relationships (relships), both may contain attributes, and attributes may be keys; in the REL metamodel, a schema may have relations, which may have again attributes.\(^1\) We only provide here an informal description of ER2REL. In a nutshell, the ER2REL transformation takes an instance of the ER metamodel as input and produces an instance of the REL metamodel following the transformation in Fig. 8.4. This transformation is described by matched rules, which are the workhorse of ATL. Matched rules define a pattern of input types and possibly a filter expression (the from-clause). Each rule is applied to each matching set of objects in the input model to create the objects in the target model that

\(^1\)For simplicity, we refer to the metamodel elements as schemas, entities, relationships, etc., instead of using schema type, entity type, etc..
Figure 8.3: ER and REL metamodels

![Diagram of ER and REL metamodels]

**Figure 8.4: The ATL transformation ER2REL**

```
module ER2REL; create UUT : REL from IN : ER;

rule S2S { from s : ER!ERSchema to t : REL!RELSchema (name <- s.name) }

rule E2R { from s : ER!Entity to t : REL!Relation (name <- s.name, schema <- s.schema) }

rule R2R { from s : ER!Relship to t : REL!Relation (name <- s.name, schema <- s.schema) }

rule EA2A { from att : ER!ERAttribute, ent : ER!Entity (att.entity=ent) to t : REL!RELAttribute (name <- att.name, isKey <- att.isKey, relation <- ent) }

rule RA2A { from att : ER!ERAttribute, rs : ER!Relship (att.relsip=r) to t : REL!RELAttribute (name <- att.name, isKey <- att.isKey, relation <- r) }

rule RA2AK { from att : ER!ERAttribute, rse : ER!RelshipEnd (att.entity=rse.entity and att.isKey=true) to t : REL!RELAttribute (name <- att.name, isKey <- att.isKey, relation <- rse.relsip) }
```

are described in the `to`-clause, assigning values to their properties (typically) based on the input objects' properties. Additional information on ATL can be found at [32, 1]. The first rule in Fig. 8.4, S2S, maps ER schemas to REL schemas, the second rule E2R maps each entity to a relation, and the third rule R2R maps each relationship to a relation. The remaining three rules generate attributes for the relations. Both, entity and relationship attributes are mapped to relation attributes (rules EA2A and RA2A). Furthermore, the key attributes of the participating entities are mapped to relation attributes as well (rule RA2AK). Notice that in the property assignment, a so-called **implicit resolution** step is needed to resolve source objects to target objects: For example the binding schema<-s.schema in E2R and R2R 'silently' replaces the ERschema value of s.schema by the RELSchema object that is created for s.schema by S2S.

In Fig. 8.5, we provide a very simple extension to the ER and REL metamodels in order to illustrate the application of our prototype to security concerns. In Fig. 8.5, the ER and the REL metamodels are extended with two additional metaclasses, i.e., one to model roles and the other to model permissions for a role-based access control policy that may be required on the ER and REL elements. Role and permission metaclasses are associated to each other and, also, permissions are associated to schemas indicating that they regulate the access to these type of resources.

In this setting, we may wonder if the security policy imposed on ER schemas is adequately translated by the transformation ER2REL (depicted in Fig. 8.6) to schemas in the target REL metamodel. For instance, let us take as a security precondition that every ERSchema can be accessed by the role 'Operator'. Is it the case that upon the execution of the transformation ER2REL it is true that every RELSchema can be accessed also by a corresponding role 'Operator'? Notice that the ER2REL (just contains two additional rules that extend the transformation ER2REL in order to map roles and permissions, its relations and assignments to resources. In our modeling language, the situation of having
operators always with granted access to ER schemas is modeled for each schema by the existence of an instance of the metaclass Role named ‘Operator’, an instance of the metaclass permission and a link between this role and permission instances, and another link between such permission and any instance of the class ERSchema. The OCL expression capturing this security precondition is:

\[
\text{inv secpre1: ERSchema.allInstances() -> forAll (x | ERRole.allInstances() -> exists (r | r.name = 'Operator' and ERPermission.allInstances() -> exists (p | r.permission -> includes(p) and p.resource -> includes(x))))}
\]

Its mapping to FOL is:

\[
\forall (x) \ (\text{ERSchema}(x) \Rightarrow \exists (r, p) \ (\text{ERRole}(r) \land (\text{name}(r) = 1) \land \text{ERPermission}(p) \land \text{permission}(r, p) \land \text{resource}(p, x)))
\]

Similarly, the OCL expression capturing the intended security postcondition is:

\[
\text{inv secpost1: RELSchema.allInstances() -> forAll (x | RELRole.allInstances() -> exists (r | r.name = 'Operator' and RELPermission.allInstances() -> exists (p | r.relpermission -> includes(r) and p.relresource -> includes(x))))}
\]

Its mapping to FOL is:

\[
\forall (x) \ (\text{RELSchema}(x) \Rightarrow \exists (r, p) \ (\text{RELRole}(r) \land (\text{name}(r) = 1) \land \text{RELPermission}(p) \land \text{relpermission}(r, p) \land \text{relresource}(p, x)))
\]

Let us next formalize this problem according to our methodology

**Definition 8.1** Let \( Q \) be the ER2RELSec ATL model transformation (it is only composed of matched rules \( \{ r_1, \ldots, r_7 \} \) and it is free from OCL recursive helper operators). Then, \( Q \) is correct with respect to precondition \( \text{secpre1} \) and postcondition \( \text{secpost1} \) if and only if, upon termination of \( Q \), the following formula is unsatisfiable:

\[
(\text{ocl2fol(secpre1)}) \land \left( \bigwedge_{j=1}^{7} \text{atl2fol}(r_j) \right) \land \neg(\text{ocl2fol(secpost1)})
\]

Once the security pre and postcondition and the rules depicted in figure 8.6 are mapped to FOL using our prototype that implements the mapping that we describe in [10], we automatically find that the formula specified above, when instantiated for our example, is always unsatisfiable as Z3 automatically shows.

\[\text{\footnotesize Notice that at this step, for simplicity reasons, we are eluding to talk about which type of access, i.e., which kind of actions (write, read, etc.) can a role perform on schemas.}\]
module ER2RELSec;
create OUT: REL from IN: ER;

rule R2R { from r: ER!ERRole
to k: REL!RELRole (name <- r.name, permission <- r.permission)}

rule Perm2Perm { from p: ER!ERPermission
to q: REL!RELPermission (name <- p.name, resource <- p.resource)}

rule S2S { from s: ER!ERSchema to t: REL!RELSchema (name <- s.name)}

rule E2E { from s: ER!Entity to t: REL!Relation (name <- s.name, schema <- s.schema)}

rule R2R { from s: ER!ERRole to t: REL!RELRole (name <- s.name, schema <- s.schema)}

rule EA2A { from att: ER!ERAttribute, ent: ER!Entity (att.entity = ent)
to t: REL!ERAttribute (name <- att.name, isKey <- att.isKey, relation <- ent)}

rule RA2A { from att: ER!ERAttribute, rs: ER!ERelship (att.relation = rs)
to t: REL!ERAttribute (name <- att.name, isKey <- att.isKey, relation <- rs)}

rule RA2AK { from att: ER!ERAttribute, rs: ER!ERelshipEnd (att.entity = rs.entity and att.isKey = true)
to t: REL!ERAttribute (name <- att.name, isKey <- att.isKey, relation <- rs.relation)}

Figure 8.6: Extension of ER2REL to map RBAC structures
9 Relation to other WPs

As mentioned in the introduction, the model-based solutions for secure software architecture developed in WP7 are meant to assist the transition between the expression of security requirements in their actual deployment on a runtime platform. In addition, some of the model-based solutions developed here, support the formal modeling of security properties that must be checked on the system’s model at runtime. These two elements explain the existence of strong interactions between some prototypes developed in this workpackage and methods and tools developed in WP6 (focusing on the expression of security requirements that must then be captured in secure models), WP8 (focusing on the development of runtime platforms on which the models developed with WP7 solutions could be deployed) and WP9 (which develops verification solutions that can be used either to verify security properties on models or to verify that the models developed with WP7 solutions are correctly implemented).

In particular, the following connections are currently established:

- Connections with WP2: tool UML4PF in this work package (see Chapter 4) has been integrated into the SDE.
- Connections with WP6: the prototype of section 2 can be used to formalize the trust and reputation model of SI* and the pattern of section 4 supports the structuring of requirements for secure information systems. The access control policies modeled in prototypes of section 7, section 6 and section 5 come from access control requirements, the properties checked by the prototype of section 8 are expressed in security requirements.
- Connections with WP8: the main relation with WP8 is in the prototype of section 6, since this tool proposes to adapt the system at runtime, according to changes in the access control policy. This tool strongly relies on reflexive middleware and dynamically adaptive platforms.
- Connections with WP9: the verification activities performed in case of model transformation (section 8) are clearly related to verification techniques that are developed in WP9. There is a strong interaction between the functionalities of Action UWE and Task 9.3.2, since both activities focus on the safe integration of security policies in Future Internet applications through a systematic transformation of policies into lower level policies that are enforceable at the implementation level of executable code.
- Connections with WP10: there are plans to use the cloud pattern (section 4) to support risk assessment for information systems deployed over a cloud infrastructure.
10 Conclusion

The different prototypes developed in WP7 during the second year of NESSoS cover a wide spectrum of security modeling activities. This spectrum reflects the different concerns that impact the security of software architecture for Future Internet systems. All prototypes are organized according to three dimensions related to model-based security design: the security concern that can be modeled or managed with the prototype; the modeling formalism(s) used to express security concerns and the system architecture; the time in development at which the prototype is meant to be used (from early model-based security design to model-driven security dynamic adaptation).

This deliverable present seven prototypes. Three of these prototypes result from the integration of techniques coming from several partners. All three collaborations have been established during the NESSoS project. In addition, there are plans for collaboration and integration for two other prototypes.

The seven prototypes presented here support the different activities included in the workpackage's tasks:

- Task 7.2 Model-based decomposition of security concerns: ActionUWE (section 7) supports modeling secure web applications through a clear separation between architecture and security (access control) concerns; the framework introduced in section 2 supports modeling trust and reputation for further integration in service-oriented architectures.

- Task 7.3 Composition and adaption of security concerns: the prototype tools presented in section 6 and section 5 support the dynamic reconfiguration of secure software architectures; the tool presented in section 8 supports the verification of security invariants in case the architecture evolves through model transformation.

- Task 7.4 Reusable architectural know-how: the prototype of section 4 supports reuse through well-defined patterns for the design of secure cloud infrastructures; section 3 introduces an original platform to reuse an experimental infrastructure for empirical investigations of model-based security design methods.

These prototypes embed initial concrete solutions to address the challenges that had been identified at the end of year one:

- The integration UML-based model-driven security has been strengthen through a joint effort from LMU, IMDEA and ETH. The transformation from UWE to ActionGUI has been precisely specified, for a complete chain from high level models to code. This collaboration will go on in year 3, to go further in the development of this tool chain.

- The prototype tools presented in section 6 and section 5 are encouraging prototypes that show the feasibility of runtime adaptation mechanisms for RBAC policies. These prototype tool chains are founded on advanced knowledge about dynamic adaptation in this domain (impact on requirements and runtime system). However, more experiments are needed to evaluate the benefits of these integrated techniques.

- Activities on reusable architecture know-how started only in year 2, leading to two prototypes already. These efforts will continue in year 3. In particular, the deployment of security architecture patterns for cloud applications will be a major challenge for the coming period.
11 NESSoS WP 7 Second-year Publications


Bibliography


[38] B. Morin, O. Barais, J. Jézéquel, F. Fleurey, and A. Solberg. Models@ run. time to support dynamic adaptation. *IEEE Computer*, 2009.


