Evaluating & Engineering: an Approach for the Development of Secure Web Applications

Marianne Busch

Dissertation
an der Fakultät für Mathematik, Informatik und Statistik
der Ludwig-Maximilians-Universität München

2016

Evaluating & Engineering: an Approach for the Development of Secure Web Applications

Dissertation
an der Fakultät für Mathematik, Informatik und Statistik
der Ludwig-Maximilians-Universität München

eingereicht von

Marianne Busch

München, den 3.6.2016

Erstgutachter: Prof. Dr. Dr. h.c. Martin Wirsing
Zweitgutachterin: Prof. Dr. Ruth Breu
Datum der mündlichen Prüfung: 28.7.2016

Eidesstattliche Versicherung

(Siehe Promotionsordnung vom 12.07.11, §8, Absatz 2 Punkt 5.)

Hiermit erkläre ich an Eidesstatt, dass die Dissertation von mir selbstständig, ohne unerlaubte Beihilfe angefertigt ist.

Marianne Busch
München, den 3.6.2016

Abstract

On a regular basis, we learn about well-known online services that have been misused or compromised by data theft. As insecure applications pose a threat to the users' privacy as well as to the image of companies and organizations, it is absolutely essential to adequately secure them from the start of the development process.

Often, reasons for vulnerable applications are related to the insufficient knowledge and experience of involved parties, such as software developers. Unfortunately, they rarely (a) have a comprehensive view of the security-related decisions that should be made, or (b) know how these decisions precisely affect the implementation. A vital decision is the selection of tools and methods that can best support a particular situation in order to shield an application from vulnerabilities. Despite of the level of security that arises from complying with security standards, both reasons inadvertently lead to software that is not secured sufficiently. This thesis tackles both problems.

Firstly, in order to know which decision should be made, it is crucial to be aware of security properties, vulnerabilities, threats, security engineering methods, notations, and tools (so-called knowledge objects). Thereby, it is not only important to know which knowledge objects exist, but also how they are related to each other and which attributes they have.

Secondly, security decisions made for web applications can have an effect on source code of various components as well as on configuration files of web servers or external protection measures like firewalls. The impact of chosen security measures (i.e., employed methods) can be documented using a modeling approach that provides web-specific modeling elements.

Our approach aims to support the conscious construction of secure web applications. Therefore, we develop modeling techniques to represent knowledge objects and to design secure web applications. Our novel conceptual framework SecEval is the foundation of this dissertation. It provides an expandable structure for classifying vulnerabilities, threats, security properties, methods, notations and tools. This structure, called Security Context model, can be instantiated to express attributes and relations, as e.g., which tools exist to support a certain method. Compared with existing approaches, we provide a finer-grained structure that considers security and adapts to the phases of the software development process. In addition to the Security Context model, we define a documentation scheme for the collection and analysis of relevant data. Apart from this domain-independent framework, we focus on secure web applications. We use SecEval's Security Context model as a basis for a novel Secure Web Applications' Ontology (SecWAO), which serves as a knowledge map. By providing a systematic overview, SecWAO supports a common understanding and supports web engineers who want to systematically specify security requirements or make security-related design decisions.

Building on our experience with SecWAO, we further extend the modeling approach UML-based Web Engineering (UWE) by means to model security aspects of web applications. We develop UWE in a way that chosen methods, such as (re)authentication, secure connections, authorization or Cross-Site-Request-Forgery prevention, can be linked to the model of a concrete web application.

In short, our approach supports software engineers throughout the software development process. It comprises (1) the conceptual framework SecEval to ease method and tool evaluation, (2) the ontology SecWAO that gives a systematic overview of web security and (3) an extension of UWE that focuses on the development of secure web applications. Various case studies and tools are presented to demonstrate the applicability and extensibility of our approach.

Zusammenfassung

Regelmäßig wird von erfolgreichen Angriffen auf Daten und Funktionen bekannter Onlinedienste berichtet. Da unsichere Anwendungen nicht nur eine Bedrohung für die Privatsphäre ihrer Nutzer, sondern auch eine Gefahr für das Image der betroffenen Unternehmen und Organisationen darstellen, ist es unverzichtbar, Anwendungen von Anfang an ausreichend zu schützen.

Zwei Gründe für unsichere Anwendungen sind, dass die Beteiligten, wie z.B. Softwareentwickler, nur selten (a) vollständig überblicken, welche sicherheitsbezogenen Entscheidungen getroffen werden müssten oder (b) wissen, welche Auswirkungen diese konkret auf die Implementierung haben. Eine kritische Entscheidung ist die Auswahl von Werkzeugen und Methoden, die in einer bestimmten Situation von Nutzen sein könnten, um die Anwendung vor Schwachstellen zu schützen. Diese Gründe führen – trotz punktuellem Schutz durch das Vorgehen nach IT-Sicherheitsstandards – ungewollt zu Software, die nicht entsprechend ihres Schutzbedarfs abgesichert ist. Die vorliegende Arbeit nimmt sich beider Probleme an.

Einerseits ist für die Entscheidungsfindung ein Verständnis von sogenannten „Wissensobjekten", wie Schwachstellen, Bedrohungen, Sicherheitseigenschaften, sicherheitsrelevanten Methoden, Notationen und Werkzeugen essentiell. Dafür ist nicht nur eine Bestandsaufnahme existierender Wissensobjekte wichtig, sondern auch deren Eigenschaften und Zusammenhänge untereinander.

Andererseits können sicherheitsrelevante Entscheidungen für Webanwendungen sowohl Auswirkungen auf Quellcodes verschiedener Softwarekomponenten haben, als auch auf Konfigurationsdateien von Webservern oder auf Schutzmaßnahmen wie Firewalls. Mit einem Modellierungsansatz, der webspezifische Modellierungselemente beinhaltet, ist es möglich Sicherheitsmaßnahmen zu dokumentieren.

Das Ziel der vorliegenden Arbeit ist es, die bewusste Absicherung sicherheitskritischer Webanwendungen zu unterstützen. Dazu werden Modellierungstechniken zur Darstellung von Wissensobjekten und zum sicheren Webanwendungsdesign entwickelt. Die Basis bildet unser konzeptionelles Framework SecEval. Es beinhaltet eine erweiterbare Struktur für Schwachstellen, Bedrohungen, Sicherheitseigenschaften, Methoden, Notationen und Werkzeuge. Diese Struktur (das sog. Kontextmodell) kann instanziiert werden, um Eigenschaften und Zusammenhänge darzustellen, z.B. Werkzeuge, die eine bestimmte Methode unterstützen. Im Vergleich zu existierenden Arbeiten wird eine detailliertere Struktur aufgebaut, die Sicherheit berücksichtigt und die Phasen des Softwareentwicklungsprozesses mit einbezieht. Zusätzlich zu dem Kontextmodell wird ein Dokumentationsschema zur Sammlung und Analyse passender Daten definiert.

Abgesehen von SecEval, das nicht domänenspezifisch ist, liegt der Fokus auf dem Bereich sicherer Webanwendungen. Genutzt wird SecEval's Kontextmodell unter anderem als Basis für die SecWAO-Ontologie – einer Art Wissenslandkarte der Webanwendungssicherheit. SecWAO bietet eine einheitliche Kommunikationsgrundlage und unterstützt Webentwickler, die systematisch Sicherheitsanforderungen spezifizieren oder Designentscheidungen treffen wollen.

Aufbauend auf der Struktur von SecWAO wird der Modellierungsansatz UML-based Web Engineering (UWE) mit Elementen zur Dokumentation von Sicherheitsaspekten erweitert. Auf diese Art können ausgewählte Methoden wie z.B. (Re)authentifikation, sichere Verbindungen, Autorisierung oder die Verhinderung von Cross-Site-Request-Forgery direkt in Bezug zur modellierten Webanwendung gesetzt werden.

Zusammengefasst unterstützt der vorgestellte Ansatz Softwareentwickler während des Entwicklungsprozesses und umfasst (1) das konzeptionelle Framework SecEval, das die Evaluation von Methoden und Werkzeugen vereinfacht, (2) die Ontologie SecWAO, die einen systematischen Überblick über Websicherheit gibt und (3) eine Erweiterung von UWE für sichere Webanwendungen. Verschiedene Fallstudien und Werkzeuge werden vorgestellt, die die Anwendbarkeit und Erweiterbarkeit des Ansatzes zu veranschaulichen.

Contents

List of Figures

Acronyms

API	Application Programming Interface
BPMN	Business Process Modeling Notation
CASE	Computer-Aided Software Engineering
CBK	Common Body of Knowledge
CPU	Central Processing Unit
CSRF	Cross-Site-Request-Forgery
CSS	Cascading Style Sheets
DOS	Denial of Service
DRM	Digital Rights Management
DSL	Domain-Specific Language
EMS	Energy Management System
FACPL	Formal Access Control Policy Language
GUI	Graphical User Interface
HSTS	HTTP Strict Transport Security
HTML	Hypertext Markup Language
http	Hypertext Transfer Protocol
https	Hypertext Transfer Protocol Secure
ID	Identifier
IDE	Integrated Development Environment
IP	Internet Protocol
IT	Information Technology
JSON	Java Script Object Notation
MAC	Mandatory Access Control
MPO	Meter Point Operator
MVC	Model-View-Controller
NESSoS	Network of Excellence on Engineering Secure Future Internet Software Services and Systems
OCL	Object Constraint Language
OWL	Web Ontology Language
pdf	Portable Document Format
PDP	Policy Decision Point
PFS	Perfect Forward Secrecy
RBAC	Role-Based Access Control
SA	Smart Appliance
SDE	Service Development Environment

SDLC	Software Development Life Cycle
SecEval	Framework for Evaluating (Security) Engineering Approaches
SecWAO	Secure Web Applications' Ontology
SNP	Secure Navigation Path
TLS	Transport Layer Security
UML	Unified Modeling Language
URL	Uniform Resource Locator
UWE	UML-based Web Engineering
XACML	eXtensible Access Control Markup Language
XMI	XML Metadata Interchange
XML	eXtensible Markup Language
XSS	Cross-Site Scripting

Danksagung

Ich möchte mich bei allen bedanken, die mich motiviert haben zu promovieren und die mich bei der Promotion unterstützt haben. Insbesondere danke ich meinem Doktorvater Martin Wirsing für die stete Unterstützung und Motivation und für die Freiheit, meinen fachlichen Interessen in der Promotion nachgehen zu können. Danke auch für die Möglichkeit über das EU Projekt NESSoS[1] Kontakte knüpfen und mein Wissen erweitern zu können. Bester Dank geht an Ruth Breu, die ich auf einer Dienstreise in Málaga ansprach und die sich an Ort und Stelle bereiterklärte meine Zweitgutachterin zu werden.

Bereits als studentische Hilfskraft am Lehrstuhl für Programmierung und Softwaretechnik (PST), arbeitete ich mit Nora Koch zusammen. Der durch sie entstandene Zugang zur Wissenschaft trug dazu bei, dass ich für das EU Projekt NESSoS und die Promotion an der Universität blieb. Besten Dank fürs Korrekturlesen hunderter Seiten, seien es Papers, Deliverables für das EU Projekt, oder die Dissertation. Danke auch für die Zusammenarbeit in NESSoS und für zahlreiche hilfreiche Diskussionen.

Sowohl für die fachliche Zusammenarbeit als auch für motivierende Gespräche möchte ich mich bei meinen Kollegen am PST-Lehrstuhl, den NESSoS Projektpartnern und besonders bei allen Mitautoren meiner wissenschaftlichen Publikationen bedanken. Danke für den Einsatz der Studentinnen und Studenten, die unter meiner Betreuung ihre Abschlussarbeiten verfassten. Viele Arbeiten waren sehr hilfreich für die vorliegende Dissertation.

Forschung ohne Ideenaustausch wäre nicht möglich, daher bin ich dankbar für all die Informationen und Gedanken, die von anderen weitergegeben wurden und für die Rückmeldung von Gutachtern. Besonderer Dank geht an die Wissenschaftler, die mir Vorabversionen ihrer Papers zukommen ließen. Ich danke allen Softwareentwicklern und IT-Sicherheitsspezialisten, die mit mir diskutiert haben und allen, die mir geholfen haben, mein Englisch zu verbessern.

Herzlicher Dank geht an meinen Freund Florian und meine Eltern, deren voller Unterstützung bei meinen Plänen ich mir stets sicher sein konnte, und an meine Freundinnen Katharina und Veronika, die – trotz der Arbeit an ihren eigenen Dissertationen – immer Zeit für mich hatten. Dankbarkeit empfinde ich auch für viele glückliche Zufälle und für ein Umfeld, in dem es möglich ist, sich mit wissenschaftlichen Fragestellungen auseinanderzusetzen.

[1]This work has partly been supported by the EU-NoE project NESSoS, GA 256980.

Part I

Introduction and Background

Chapter 1

Introduction

"Cybercrime is a growth industry. The returns are great, and the risks are low" [50, p.2]. This means that around the world, much effort is spent on finding and exploiting vulnerabilities in the user's behavior as well as in software. Although the reported percentage of web applications containing vulnerabilities varies according to the type of measurement, it indicates that more than three out of four web applications are vulnerable,[1] which alarmingly often results in disclosing confidential data or in systems getting out of control.[2]

1.1 Motivation

Technical vulnerabilities may arise from misunderstandings within a development team, from lack of knowledge, from losing track of complex coherences,[3] from unclear responsibilities[4], from a lack of risk awareness[5], from security by obscurity,[6] or from arbitrary careless mistakes that are made during the development process. Pressure of time and competition urges all actors to focus on functional aspects, i.e., to provide as many useful features as fast as possible. Security aspects are often categorized as non-functional requirements, as actually vulnerable software can successfully complete its tasks. Even a compromised application might offer its functionality to unsuspecting users for many years, while leaking data.

It is commonly known that errors made in early phases of the development process can become expensive, as they are in many cases difficult to correct in implemented software systems [171, p. 3]. As a consequence, it is common practice for companies to establish organization-wide policies for secure software development. However, the resulting documentation tends to be

[1]Web applications with vulnerabilities: 96% [51, p.3], 76% [210, p.38] of those that were scanned.

[2]Critical vulnerabilities: 20% [210, p.38] of all vulnerabilities discovered, 46% websites of those that were scanned have high security vulnerabilities [2, p.5].

[3]"Humans get bored, hungry, tired, hung-over, or otherwise distracted and that can introduce bugs into the web application being developed" [171, p. 2].

[4]In practice, one can still hear "I do not have to care about security, because our security engineers / our server administrators / the used web framework / our firewall will somehow 'secure' it."

[5]Securing software is demanding and can be an obstacle to other requirements (that are often easier to realize), which might lead to devaluating this goal to resolve cognitive dissonance: "Security is not that important for us! Nobody could ever be interested in what we do / in our data / in our system!"

[6]Similarly: "Nobody will ever guess right / recognize this!" Further myths can be found in [57].

lengthy, which makes it difficult to get a comprehensive view. Parts of these policies require to turn high-level directives into appropriate design decisions for concrete applications, which can be a complex task; especially if appropriate methods or tools[7] have to be chosen that support the directives. In addition, due to deadline constraints, developers frequently postpone reading or updating comprehensive documentation, which can be fatal in cases of personnel changes.

The exploit of a vulnerability can damage the reputation of the manufacturers of software, as well as the reputation of those who employ it. Common consequences are corporate espionage or financial damage due to lost confidence of customers that found their private data disclosed [92, 174]. In some cases, misuse of systems can lead to legal liability. In other cases, vulnerabilities can even threaten product safety, as e.g., some functionality of building automation systems, medical devices and automotive software are accessible via web front-ends. In the end, a single vulnerability can be enough to compromise an application.

Unfortunately, current engineering approaches are neither powerful enough nor used diligently enough to secure web applications[8] in a way that the number of vulnerabilities drops significantly in practice. Many security methods exist, as e.g., the ISO/IEC 27000 [105] family, the Microsoft SDL [124], or SABSA [196]. Most of them try to reduce security to simple lists of keywords and guidance, but in reality interdependencies and levels of abstraction are more complex and interconnected. Thus, checklists are better than nothing, but easily lead to applications that are only secured selectively.

1.2 Aim

The main goal of this dissertation is to support the construction of secure applications – in particular web applications – by facilitating conscious decision making, method evaluation and security documentation. Therefore, we aim at providing (a) a conceptual framework for general IT security concepts, (b) a knowledge map (ontology) of web application security concepts and (c) a graphical representation for security concepts that are employed in a concrete web application. These three levels of abstraction are depicted in figure 1.1. In short, *a* should support evaluation, *b* should document how concepts of web application security are interrelated in general, and *c* should be suitable for documenting secure web applications' requirements and designs. For constructing a secure web application, "facilitating conscious decision making" matters especially at the transitions between these levels: first, when evaluating which methods and tools should be used for the development and second, when determining an application's design.

The most abstract level we consider is the level of general IT security concepts, such as assets, methods, notations, tools, security properties, vulnerabilities, or threats. These concepts are referred to as "knowledge objects". Software developers, architects, security engineers and other involved parties (henceforth called "practitioners", if not specified more precisely) have to understand and evaluate knowledge objects they work with in order to make rational decisions. For example, practitioners need to choose appropriate methods and tools, as e.g., libraries for access control that assist in securing their applications. As evaluations are time consuming, we want to be able to document research questions[9] and corresponding results. As a consequence, reusing

[7]A method is a strategy to tackle a problem; a tool usually refers to software that supports a method.

[8]The term *web application* stands for a "web-based software that performs actions (functionality) based on user input and usually interacts with back-end systems" [171, p. 2].

[9]Note that these research questions do not necessarily refer to scientific research.

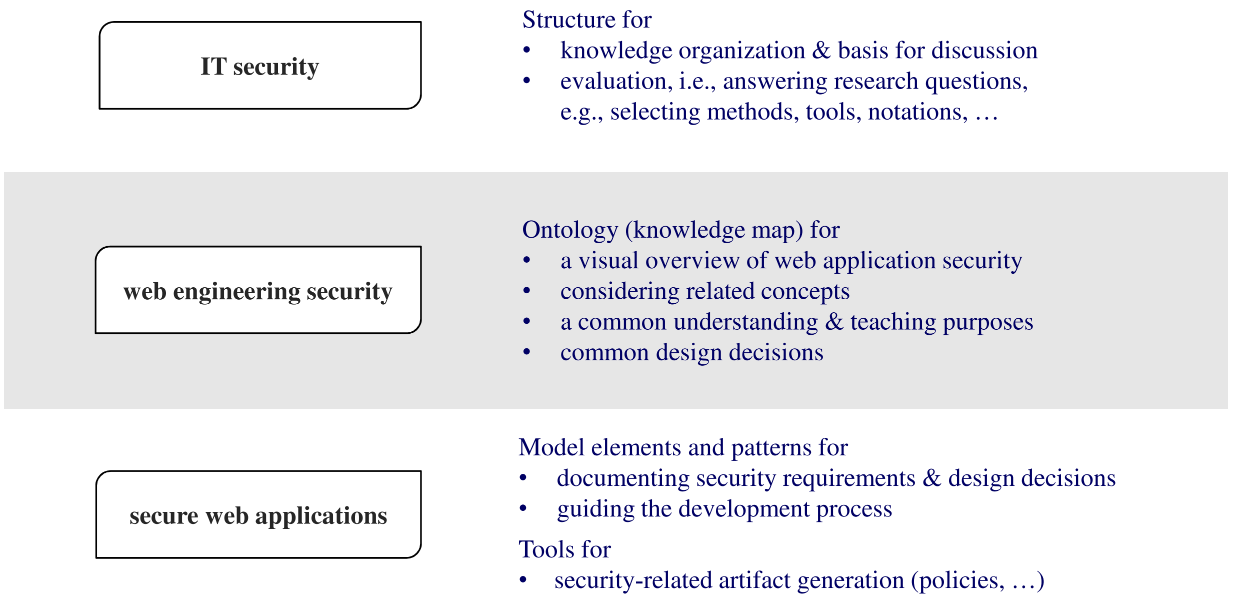

IT security
Structure for
• knowledge organization & basis for discussion
• evaluation, i.e., answering research questions, e.g., selecting methods, tools, notations, …
web engineering security
Ontology (knowledge map) for
• a visual overview of web application security
• considering related concepts
• a common understanding & teaching purposes
• common design decisions
secure web applications
Model elements and patterns for
• documenting security requirements & design decisions
• guiding the development process
Tools for
• security-related artifact generation (policies, …)

Figure 1.1: Aims of our approach

data from a resulting knowledge base becomes easy. Contrary to existing evaluation approaches, the conceptual evaluation framework we propose should allow to represent methods from different areas appropriately, as e.g., design methods versus test methods and make relationships explicit that have not been considered in existing approaches.

From the teaching experience of the author and from discussions with members of the security community, we derive that students as well as practitioners struggle to get a comprehensive view of concepts and their relations, especially in the vast area of web engineering security. Even in security books in this domain, relationships are missing, as they are often structured according to buzzwords, rather than according to a knowledge map or ontology[10] that systematically connects concepts. Fortunately, ontologies can be created by instantiating the knowledge objects mentioned above, which leads to a second level of abstraction (cf. figure 1.1). At this level, concrete security concepts are located, as e.g., Cross-Site Scripting (XSS)[11] (a kind of vulnerability) or authorization (a kind of method). We aim at building a secure web applications' ontology that provides a comprehensive view of web engineering security so that it is easy to consider related concepts and to come to an understanding that can be shared with others. The ontology should allow deriving where decisions have to be made deliberately to keep common risks under control.

The overall objective is to secure web applications and data they access. Company-wide security risk management can help to pursue this target, while in the end, the technically most decisive factor is the reasonably secure implementation and protection of applications and systems. This

[10]"An ontology is a formal explicit specification of a shared conceptualization for a domain of interest" [208]. This means, an ontology is a knowledge map that structures and relates concepts based on a predefined, formal structure. (In addition to an informal knowledge map, a knowledge base provides structured information about concepts it contains. We do not differentiate between the terms ontology and taxonomy.)

[11]XSS is a kind of injection that aims at adding malicious script code – usually JavaScript – to a website so that the browser executes the code [195, p.24].

makes it necessary to understand complex coherences on the level of code with its technicalities like missing bound checks or unsafe memory functions, as well as on the level of design. Common design goals are, e.g., shielding the internal part of a web application from unauthorized access, preventing XSS attacks or enforcing reauthentication after a while. Thereby, the challenge is to consider security properties at the appropriate moment during the development process. This third level is depicted at the bottom of figure 1.1, our goal is to provide a specification language for designing secure web applications. This approach should comprise elements and patterns for documenting security requirements and design decisions graphically or textually and provide tool support for security-related artifact generation. So far, approaches that take security into account are restricted to simplified web applications and to the best of our knowledge, no approach deals with security goals that are related to common web-specific vulnerabilities.
In short, our research questions are the following:

RQ1 Which key concepts emerge while developing secure software? Which properties do these concepts have and how are they related to each other? How can this knowledge be applied to evaluate concepts?

RQ2 How are assets, security engineering methods, notations, tools, security properties, vulnerabilities and threats related in the domain of web application security?

RQ3 How can security aspects of web applications be expressed? How can resulting models be used in the development process?

1.3 Approach

In order to answer the aforementioned questions, our approach consists of three interconnected parts. According to the levels presented in figure 1.1, we present a novel conceptual evaluation framework, called SecEval, a Secure Web Applications' Ontology (SecWAO), and security extensions for a notation that allows to model web applications, namely UML-based Web Engineering (UWE) [224, 116]. Figure 1.2a depicts the parts of our approach.

SecEval provides an expandable structure for knowledge objects, such as assets, methods, tools, vulnerabilities, and security properties (cf. top of figure 1.2b). This model can be instantiated to express attributes and relations, as e.g., which tools exist to support a certain method. The top of figure 1.2b exemplarily shows a UML class diagram with attributes for the concept of a `Tool`, as e.g., the license software is distributed under. Some associations are denoted by line linkings to express facts like "a method is related to a vulnerability". For example, a method can detect a vulnerability or shield it from being exploited. Generally, SecEval's model can also be used to describe tools and methods that are not security-related. In addition, we define a documentation scheme for the collection and analysis of information. For validating the soundness of SecEval, SecEval's model is applied to a case study of web vulnerability scanning and it is used as a basis for the novel Secure Web Applications' Ontology (SecWAO).

In the terminology of ontologies, SecWAO is the ABox (assertion component) that provides concrete data for the TBox (terminological component) provided by SecEval. From a modeling perspective this means that SecWAO instantiates SecEval's model. The center of figure 1.2b depicts an excerpt of SecWAO as a UML instance diagram. The colors refer to the SecEval

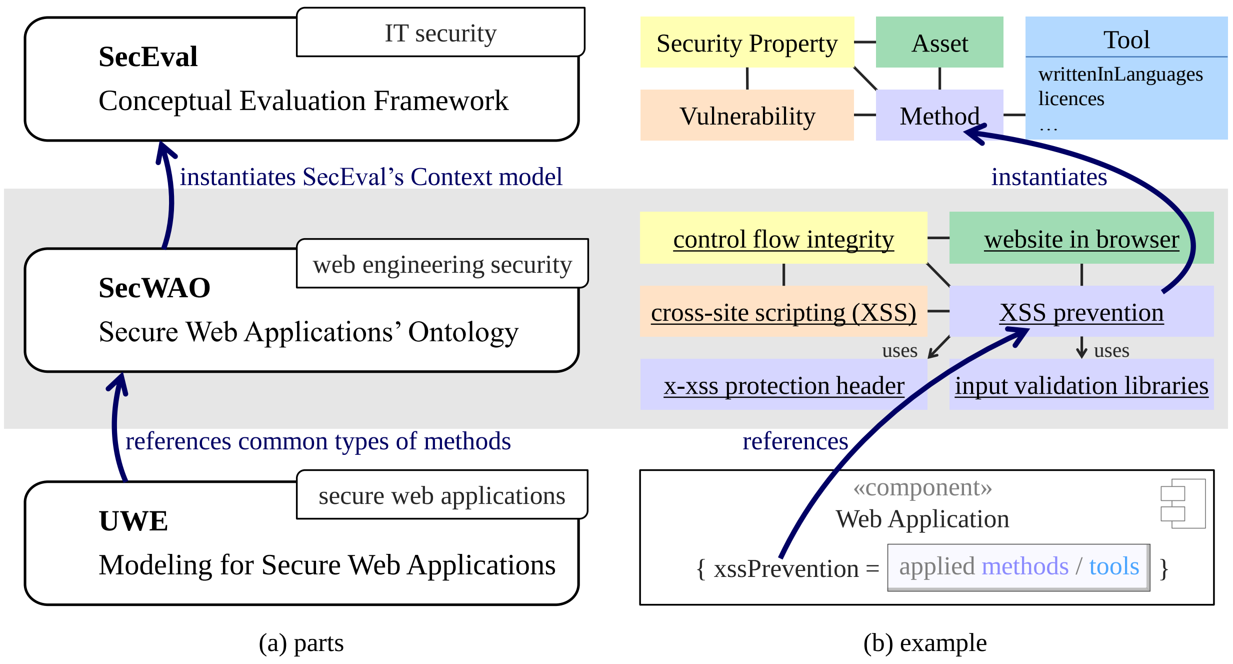


Figure 1.2: Construction of our approach

elements, e.g., control flow integrity[12] is a security property and XSS prevention is a method to prevent an asset like a website from containing XSS vulnerabilities. Practitioners can see at a glance that two common methods to be considered for XSS prevention are the x-xss protection header (a http protocol header) and the use of input validation libraries.

Ideally, practitioners document or plan how a web application already is or should be configured or implemented. To support this task, we extend the modeling approach UML-based Web Engineering (UWE) by means to model security aspects of web applications. As a consequence, security properties and methods can be represented, as e.g., authentication, reauthentication, secure connections, authorization, user zones, Cross-Site-Request-Forgery (CSRF) prevention, under attack mode and injection prevention. The bottom of figure 1.2 shows that UWE references common types of methods from SecWAO, such as XSS prevention, which is specified as a UML tag. The tag indicates that the risk of XSS attacks to the "Web Application" component is reduced by selected methods or tools.

As UWE is provided as a UML profile that defines stereotypes with tags and common patterns, modeling with UWE is possible with ordinary UML editors. Moreover, we facilitate web engineering with several proof-of-concept tools, e.g., for abbreviating common modeling tasks. Our approach is partly model-driven with respect to security features, as various artifacts, like configuration files or access control policies can be derived from UWE models. An example for a configuration is the restriction of a web user's navigation through a web application to a predefined path.

Among other case studies, we apply the extended UWE approach to a Smart Home case study, defined by Siemens. In this case study, a server is installed in the home of a SmartGrid customer.

[12] Control flow integrity is the property of software that restricts a user to execute functions in a predefined order, according to the program logic.

The web application deployed at this server – namely the Energy Management System (EMS) – displays energy consumption and enables the customer to trade energy or to control home appliances. With UWE we are able to design different views of the application, i.e. to highlight essential security requirements and methods in the context in which they are relevant.

1.4 Usage Example

A basic usage scenario that encompasses all layers of our approach is depicted in figure 1.3. At least since having a look at the excerpt of SecWAO, practitioners know that input validation libraries can be used for XSS prevention. In an early phase of the development process, the decision to use input validation libraries can be documented using UWE's tag {xssPrevention = "input validation libraries"}.

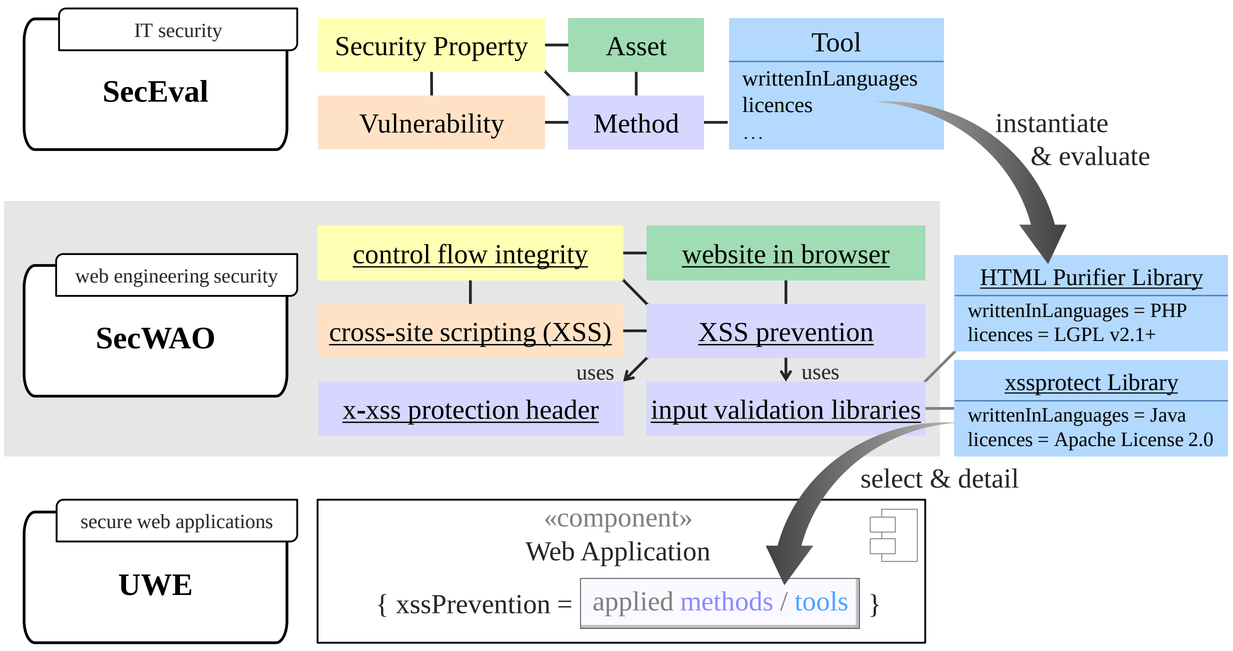


Figure 1.3: Usage example

Searching the web shows that many libraries are worth considering. Practitioners can instantiate SecEval's model to be able to evaluate a selection of tools. The tools can also be connected to existing knowledge, as e.g., to the concepts of SecWAO. Given that a team of practitioners develop using Java and their evaluation suggests that using the xssprotect Library[13] meets their requirements best, they can refine their UWE model to {xssPrevention = "xssprotect Library"}. Consequently, future developers can see at one glance which means of protection are applied to certain components. Note that this example is an oversimplification, as several UWE views exist and the UWE's Content model that is used to represent components is just one of them. Other models are, e.g., describing the user's navigation possibilities within a web application or specify access control rights on functionality or data.

[13]xssprotect. https://code.google.com/p/xssprotect/

Most Software Development Life Cycles (SDLCs) contain requirement elicitation tasks as well as design, implementation, testing and maintenance tasks. As the chronological order of these tasks is irrelevant for our approach, we use the terms "SDLC" and "development process" interchangeably. Thus, we do not restrict ourselves to a certain paradigm or technical framework, but try to model security in a way that raises comprehension and awareness so that "security" becomes a first-class citizen.

1.5 Outline

The next chapter completes part I by introducing existing approaches and common terms that are related to our work. In addition, gaps are discussed. The remainder of this thesis is structured as follows, whereby each chapter in part II and part III closes with a section that comprises a summary which also revisits major related work:

Part II Evaluating and Relating Security Concepts

Chapter 3 presents our novel conceptual framework SecEval that (a) considers the development phases a tool or method is used in and (b) is more expressive than existing approaches with regard to its relations, attributes and process support.

Section 3.1 introduces a common procedure for answering research questions that are related to knowledge objects.

Section 3.2 describes SecEval's architecture. On the one hand, SecEval shows the relations between security engineering methods, notations, tools, assets, security properties, vulnerabilities and threats. Features and relations of these elements are represented in a pattern-based structure. On the other hand, it describes artifacts in the process of data collection and data analysis and their interrelations. An extension of SecEval for existing risk rating and method evaluation approaches concludes this section.

Section 3.3 reports on a guided interview that we executed to find out whether SecEval's model meets the needs of researchers from various areas.

Section 3.4 examines whether SecEval is sound, easy to use and expressive enough to be used in a case study that evaluates web vulnerability scanning tools.

Chapter 4 Our SecWAO ontology describes how concrete security engineering methods, notations, tools, security properties, vulnerabilities and threats are related in the domain of web application security. Besides, SecWAO demonstrates that SecEval can be used as a basis for an ontology.

Section 4.1 introduces SecWAO by example.

Section 4.2 shows an excerpt of SecWAO's security properties.

Section 4.3 relates common methods for engineering and securing web applications.

Section 4.4 structures vulnerabilities and links them to threats.

Section 4.5 discusses important features for an implementation of a knowledge base that manages SecEval's models. Our prototype can document evaluation results and describe and connect concepts of SecWAO.

Part III Engineering Secure Web Applications

Chapter 5 gives an idea of our case studies and of UWE's model types, before presenting modeling elements that are able to express required security properties and design decisions in UWE respectively in UML.

Section 5.1 gives an overview of the case studies we use to demonstrate how different aspects of web applications can be modeled.

Section 5.2 introduces frequently used models of the modeling language UWE.

Section 5.3 presents novel modeling elements that express security[14] aspects of web applications.

Chapter 6 examines artifacts that can be generated from UWE models and the purposes these artifacts serve.

Section 6.1 experiments with a textual representation of UWE models in a Domain-Specific Language (DSL) written in Scala [155].

Section 6.2 presents a way to generate access policies (XACML [149]) and to transform them to a more formal representation, namely FACPL [130].

Section 6.3 presents tools that can be used in a row for modeling access control and testing XACML policies.

Section 6.4 introduces the transformation of a high-level language as UWE into a modeling language like ActionGUI [12] that uses detailed models that substitute code for restricted web applications.

Section 6.5 demonstrates that information from UWE models can be used to secure the navigation flow of a web application, i.e., disallowing users to leave predefined paths.

Part IV concludes with a summary and an outlook.

An appendix contains additional information, e.g., about the UWE profile and our EMS case study. In the end, the author's list of publications and the bibliography finalize this thesis. Thereby, the author's list of publications includes references to chapters of this thesis that cite the author's previous publications and statements about her contribution to joint works.

[14] Although "(IT) security" is often called "Cybersecurity" when the Internet is concerned, we stick to the former, because the latter is increasingly used as a buzzword.

Chapter 2

Background

In this chapter, we provide background information on existing approaches in the field of secure (web) application development. Section 2.1 introduces current efforts and standards to improve secure development processes, section 2.2 is devoted to ontologies, knowledge bases and evaluation approaches, and in section 2.3 we briefly present approaches for modeling secure web applications. This chapter contains and extends parts of all "related work" sections from the author's publications (listed on pages 177f.). Please note that details regarding similarities or differences to our work will be discussed in following chapters of this thesis and summarized at the end of each chapter. In addition, chapter 6 provides specific background in the area of different artifact generation approaches.

Although we learned about background and related work over time, we tried to systematically broaden our knowledge about security ontologies and secure web application modeling approaches, using mainly combinations of the following three sets of search terms, choosing zero or more terms of each set: {model, modeling, model-driven, aspect-oriented, requirements, designing, developing, engineering}, {secure web, web security, security}, {applications, access control, RBAC, life cycle, ontology, taxonomy, evaluation, concept, knowledge base}. We searched the web using the resulting search terms to find relevant papers, books and websites and skimmed through at least the first ten search results. Main resources were Google Scholar[1], Springer[2], IEEE Computer Society[3], ACM Digital Library[4] and various web security books. Our strategy included looking into sources, categorizing seemingly interesting ones using JabRef[5] and following interesting references to (a) cited literature, (b) other work from the same author or (c) other work from the same conference, if available.

2.1 Secure Development Processes and Standards

Incorporating security into development processes means to add activities to ensure security features in every phase of the Software Development Life Cycle (SDLC). These approaches

[1]Google Scholar. http://scholar.google.com/
[2]Springer. http://springerlink.com/, http://www.springerprofessional.de/
[3]IEEE Computer Society. http://www.computer.org/
[4]ACM Digital Library. http://dl.acm.org/
[5]JabRef. http://sourceforge.net/projects/jabref/

enrich the software development process by, e.g., security requirements, risk assessment, threat models during software design, best practices for coding, the use of static analysis code-scanning tools during implementation, and the realization of code reviews and security testing.

Many companies define their own secure development process in order to ensure that the software they develop has as few vulnerabilities as possible. A contribution in this area is the Microsoft Security Development Lifecycle (SDL) [124]. It is a software development process used to reduce the software maintenance cost and to increase the reliability of software concerning software security-related bugs. The Microsoft SDL can be adapted for the use in a classical waterfall model, a spiral model, or an agile model. Other companies also define their security SDLC, as e.g., Cisco SDL Process[6] or SAP's Product Innovation Lifecycle[7]. In addition to general development processes, Glisson worked out a development process that focuses on web engineering [89]. For our work, the concrete phases or activities of a SDLC and their order is less important than general security awareness. Therefore, our approach does not specify a detailed process, but should be used in an agile way.

In addition to SDLCs, security standards, as the ISO/IEC 27000 family [105] can be applied (e.g., ISO/IEC 27034 deals with application security management[8]). They go beyond software development and define an information security management system that requires the specification of security guidelines for policies, processes and systems within an organization. Important is that most standards do not define how to increase security, but they define areas that have to be taken into consideration in order to create meaningful security guidelines. This is related to our aim of conscious decision making during the SDLC. Similar documents are released by NIST's Computer Security Division[9] and the German Federal Office for Information Security (BSI)[10]. For example, in SP 800-64 Rev.2, NIST details "Security Considerations in the System Development Life Cycle" by describing main SDLC phases with their expected outputs and typical security roles within software projects.

All three, the ISO/IEC 27000 family, the NIST and the BSI documents, aim at providing comprehensive information about the development and maintenance of secure software and systems. Included information is structured according to the tree of headings in the documents, but no underlying structure is used to interconnect mentioned concepts and advices. However, for the Common Criteria for Information Technology Security Evaluation [59], a standard for evaluating whether software products fulfill specified security properties, Białas [23, 24] structures and relates terms that are specific for the Common Criteria.

In practice, search engines like ITU's approved ICT security standards database[11] can assist in finding appropriate information within standards. For further information about IT security standards, the interested reader is referred to Wikipedia[12].

[6]Cisco SDL Process. http://www.cisco.com/web/about/security/cspo/csdl/process.html

[7]SAP - Embedding Security in the Product Innovation Lifecycle. https://support.sap.com/content/dam/library/support/support-programs-services/support-services/sec-sw-dev.pdf

[8]ISO/IEC 27034. http://www.iso27001security.com/html/27034.html

[9]NIST Computer Security Resource Center. http://csrc.nist.gov/publications/PubsSPs.html

[10]BSI-Standards zur Internet-Sicherheit (German). https://www.bsi.bund.de/DE/Themen/Cyber-Sicherheit/ISi-Reihe/ISi-Reihe_node.html

[11]ITU's approved ICT security standards database https://www.itu.int/ITU-T/security, described in further detail in http://www.itu.int/en/ITU-T/studygroups/2013-2016/17/ict/Pages/ict-part02.aspx

[12]Wikipedia: Cyber security standards. https://en.wikipedia.org/wiki/Cyber_Security_Standards

At the enterprise level, SABSA (Sherwood Applied Business Security Architecture) [196] aims to be an "overarching framework that binds [existing methods, models and standards] all together into a single holistic view of how to design and manage enterprise security"[13]. Thus, SABSA is a framework that tries to establish security concerns as a central part of enterprise management.

Another approach is to train and to certify security engineers according to many programs, as e.g., (ISC)^2's[14] Certified Information Systems Security Professional (CISSP) or Certified Secure Software Lifecycle Professional (CSSLP). In contrast, for developers, rather few security-related certificates seem yet to exist[15,16], which strengthens the feeling that security is often seen as something that should be taken care of by "someone else", instead of by developers themselves.

For general software development, the 25 most dangerous software errors are presented by CWE/SANS[17]. An example for supporting in particular the secure development of web applications along the SDLC is the Open Web Application Security Project (OWASP). It comprises, beyond others, a set of guides for web security requirements, cheat sheets, a development guide, a code review and testing guide, an application security verification standard (ASVS), a risk rating methodology, various tools and a Top 10 of privacy risks as well as a Top 10 of web security vulnerabilities [161].

2.2 Ontologies, Knowledge Bases and Evaluation Approaches

In this section, we introduce terms (section 2.2.1) and existing security-related ontologies, knowledge bases and evaluation approaches (section 2.2.2).

2.2.1 Terms

As already written in the introduction, "an ONTOLOGY is a formal explicit specification of a shared conceptualization for a domain of interest" [208]. An ontology usually comprises an ABox (assertion component), i.e. the concrete elements, and a TBox (terminological component) that defines underlying concepts. In this thesis, we usually refer to the ABox when using the term "ontology". Some knowledge bases employ ontologies as a TBox structure for the knowledge objects they contain, as e.g., [227, 121, 49]. This underlines Fensel's statement that "providing shared domain structures is becoming essential, and ontologies will therefore become a key asset in describing the structure and semantics of information exchange" [82, p. 3]. We agree with his estimation and want to add that insights of how things are related have been fascinating humans since ancient times (cf. ontology in philosophy).

A KNOWLEDGE BASE is "a technology used to store complex structured and unstructured information used by a computer system."[18] In our work, we are interested in structured infor-

[13]SABSA FAQ. http://www.sabsa.org/faq-page

[14](ISC)^2 Certifications. https://www.isc2.org/credentials/default.aspx

[15]GIAC. Security Certifications: Software Security. https://www.giac.org/certifications/software-security

[16]EC-Council Certified Secure Programmer (ECSP).
http://www.eccouncil.org/Certification/ec-council-certified-secure-programmer

[17]CWE/SANS Top 25 most dangerous software errors. https://cwe.mitre.org/top25/

[18]Wikipedia: Knowledge Base. https://en.wikipedia.org/wiki/Knowledge_base

mation and in information that can not only be used by a computer system, but also directly by humans. In Software Engineering, a shared understanding is especially helpful when large teams work together and experts are recruited for varying periods of time. Knowledge bases aim at supporting software and security engineers to learn about the state-of-the-art to avoid "reinventing the wheel" [80, p.2]. Therefore, knowledge bases can, e.g., be used to structure information for evaluation purposes, such as the evaluation of different offers or the selection of a software tool to work with.

Knowledge bases are often implemented as web applications (e.g., using semantic Wikis, such as the Semantic MediaWiki [120]) and if they are structured, they can be used similarly to ontologies that are accessible online, such as WebProtégé [206]. This means that collaborative viewing and editing of elements contained in the knowledge base or ontology is possible. Technically, ABoxes and TBoxes can be specified using the Web Ontology Language (OWL), which is supported by many ontology editors and common in the field of semantic web. However, we think that most software engineers are more familiar with UML class diagrams, which can express concepts, relations between concepts and concepts' attributes. As this is sufficient, we decided in favor of UML for our SecEval and SecWAO approach, although contained data could easily be exported to OWL.

The underlying EVALUATION APPROACH, our SecEval approach is built upon, is the so-called "Systematic Literature Review" of KITCHENHAM et al. [113], which seems to be the main evaluation approach used in software engineering. Their aim is to answer research questions by systematically searching and extracting knowledge of existing literature. We go even further using arbitrary resources in addition to available literature, such as source code or experiments that are carried out to answer a research question. A systematic literature review is executed in three main steps: first, the review is planned, then it is conducted and finally results are reported.

2.2.2 Existing Approaches

The COMMON BODY OF KNOWLEDGE (CBK) [15] defines a model to collect and describe methods, techniques, notations, tools and standards and also encompasses Kitchenham's approach. CBK's techniques do not specify activities (process steps) for applying them, as methods do. The CBK is implemented as a semantic Wiki [49] and serves as a knowledge base to which queries can be posted. As we used the CBK in the EU project NESSoS [143] (and were involved in the initial brainstorming process as well as in further discussions), it can be seen as a starting point for our SecEval approach.

INCAMI (Information Need, Concept model, Attribute, Metric and Indicator) [156] is a conceptual framework that specifies general concepts and relationships for the quality evaluation of a web application. The model can be used to represent a tree of requirements a web application should fulfil and relate requirements with measurement and evaluation concepts. The focus is not on comparing different applications, but on measuring the quality of various requirements within an application so that its quality can be improved. INCAMI concepts and relationships are specified as a UML class diagram. We also stick to use UML for graphical representation of our SecEval approach and implemented separation of concerns through UML packages.

MOODY [136] proposes an evaluation approach which is based on experiments and centers the attention on practitioners' acceptance of a method, i.e. its pragmatic success, which is defined as "the efficiency and effectiveness with which a method achieves its objectives". Efficiency is related to the effort required to complete a task and effectiveness to the quality of the result. Practitioners are invited to use methods and afterwards answer questions about perceived ease of use, perceived usefulness and intention to use. This approach is integrated as an extension of SecEval (cf. section 3.2.4).

Regarding web security, Salini and Kanmani [182] present an approach for creating an ontology of security requirements, including the concepts of assets, web applications, vulnerabilities, threats, security requirements and stakeholders. Although they define relations between these concepts, they do not provide attributes to describe them, as e.g., SecEval does. They mention some examples for these concepts, but do not provide a concrete ontology. The reader is referred to their related work section for a chronological list of security-related ontologies.

In [69, 68] Denker et al. provide examples of elements and their relations for secure web services. In contrast to SecWAO, their ontology is restricted to a tree of subclasses and properties of the concepts “Credential” and “SecurityMechanism”. They use a high level of abstraction and due the lack of different concepts, they e.g., group “authentication, authorization, access control, data integrity, confidentiality, privacy, exposure control, anonymity, negotiation, policy, key distribution” inside of “SecurityNotation”, which itself is a descendant of “SecurityMechanism”. In SecEval and SecWAO, we differentiate between security properties like confidentiality, methods like authentication and assets like credentials.

Besides the CBK, another approach with teaching purposes regarding security is called Cyber Security Learning by security Ontology Browsing (SLOB). This project seems to be abandoned, as few information is available and the prototype[19] does not fully load. In addition, the same working group [56] provides an editor for security ontologies, called Security Ontology eXpert tool (SOX)[20]. In SOX, an attribute that can be specified for describing a concept is, e.g., an excerpt taken from an explanation in a book. The approach to enrich an existing ontology by exploring textbook indexes is described in [222].

The OWL ontology enriched in [222] is presented by Herzog et al. in [101]. It primarily focuses on the classification of assets, threats, vulnerabilities and countermeasures and also contains some web-related threats, as e.g., Cross-Site Scripting (XSS). However, it lacks common vulnerabilities of web applications, as cross-site-request forgery, clickjacking or methods for session management (cf. chapter 4).

In [83], Fenz and Ekelhart introduce an ontology based on OWL that comprises (among others) assets, security attributes, threats, vulnerabilities and controls. Additionally, they use formal axioms to test if all necessary information to describe a concept is specified in a concrete ontology. Unlike SecWAO, their ontology is rather abstract; even so they also include physical security, as e.g., dumpster diving, safety doors or smoke detectors.

Kim et al. [112] use OWL to create an ontology that consists of several parts, like main security, credentials, algorithms, assurance or semantic web services’ security. The main ontology includes security objectives and security concepts like protocols, mechanisms (e.g., for securing a network) or policies. Aside from cookies and some Internet protocols, web application security is not mentioned. Unfortunately, the server hosting the full ontology files seems no longer to be online. In contrast to SecEval and the CBK, neither [101], [83] nor [112] allow for the representation of methods that are not directly related to security, as our SecEval approach does.

[19]SLOB. http://cis.csi.cuny.edu:8080/SLOB/
[20]SOX. http://cis.csi.cuny.edu/~project/SKATClient/

Structures of ontologies can also be used as a basis for eliciting security requirements, as already seen in Salini and Kanmani [182]. The i* metamodel [180] is the basis of a vulnerability-centric requirements engineering framework introduced by Elahi et al. [78]. This extended, vulnerability-centric i* metamodel aims at analyzing security attacks, countermeasures, and requirements based on vulnerabilities. Similar to our approach, the metamodel is represented using UML class models. Instances of this metamodel use an i*-specific representation, which is not based on UML. Main elements of the metamodel are: vulnerability, attack (which can exploit a vulnerability; executed by an actor), effect and security impact (e.g., on a resource). Although the terms ontology or taxonomy are not used in [78], Elahi et al. provide a detailed example of a browser and a web server, including threats by a hacker and a fake web site. For this example they analyze countermeasures for a concrete system in order to estimate the risk of vulnerability exploitation. As they do not provide an ontology, our SecWAO could be used to enable security engineers to systematically examine relevant security mechanisms of web applications when modeling with i*.

Another approach that focuses on vulnerabilities is described by Wang et al. [223]. Their concept model is less detailed than the i* metamodel. They create a knowledge base that can be queried using the Semantic Web Rule Language (SWRL) [179], as unlike our approach, they do not use graphical models.

One year after SecEval was published, Souag et al. developed a quite similar ontology (ABox-only) for eliciting security requirements [203]. It is divided in three parts: treatment (which mechanism can help to fulfill a security requirement), risk (threats, risks and vulnerabilities) and organization (asset, location, organization, person). Tools like the Adamant framework[21] also include persons, to be able to assign security requirements to them. Their realization in concrete systems can then semi-automatically be tracked.

In practice, several databases exist that collect data about vulnerabilities and threats. For example, the MITRE corporation maintains CAPEC [212], a threat database; CVE [213], a database for common vulnerabilities; and a weakness database, namely CWE, which stands for Common Weakness Enumeration. Weaknesses are "a type of mistake in software that, in proper conditions, could contribute to the introduction of vulnerabilities within that software."[22] For example, XSS is a CWE weakness[23] that can be used for a CAPEC attack that includes "Scripts in existing Scripts"[24]. CVE references that a XSS vulnerability was located in former versions of the wiki software MediaWiki[25,26]. As a weakness can be seen as a generalization of a vulnerability, we stick to the latter term, as further discussed in section 3.2.1. The database structure (TBox) of CAPEC, CVE, and CWE can be downloaded as XML Schema Definition (XSD) files. Everybody can join the community to suggest potential entries, which leads to huge knowledge bases with many detailed practical examples. In addition to these databases, MITRE develops the Structured Language for Cyber Threat Intelligence Information (STIX) [214], which provides a kind of ontology to describe campaigns, adversary tactics, techniques, and procedures

[21]Adamant. http://adamant.q-e.at
[22]CWE Glossary - Weakness. https://cwe.mitre.org/documents/glossary/
[23]CWE-79. http://cwe.mitre.org/data/definitions/79.html
[24]CAPEC-19. http://capec.mitre.org/data/definitions/19.html
[25]CVE-2008-5249. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-5249
[26]MediaWiki. https://www.mediawiki.org

as well as incidents. An overview of malware-related languages and ontologies is presented by Obrst et al. [153].

Often, it is a good idea to reference item IDs from CAPEC, CVE, CWE or the OWASP Top 10 when referencing vulnerabilities and threats, as e.g., can be done in ontologies like SecWAO. However, for SecEval, we take a step back as we are not only interested in vulnerabilities and threats that exploit vulnerabilities, but also in the interconnections of security properties, assets, methods, notations and tools. In addition, we make use of inheritance of attributes[27] (top-down), instead of pointing to generic blocks of information, as used in CAPEC's, CVE's, and CWE's schema definitions (bottom-up [148]). Both strategies have their advantages and disadvantages: Bottom-up approaches are easier to scale due to modularization and top-down approaches facilitate focusing on relations and context-sensitive attributes due to their fundamental element structure, which is valuable for evaluation and teaching purposes. For example, the CBK also uses the top-down approach, which makes it easy to define attributes that should be available for all subtypes of the most general "KnowledgeObject" type.

Some ontology structures, like the HL7's Security and Privacy Ontology [5] and the ObjectSecurity's PrivacyOntology [123] seem mostly to be used for EXCHANGING DATA in a compatible way. HL7 is a set of standards for healthcare that defines the exchange of health-related data. The PrivacyOntology was invented by a company called ObjectSecurity to specify access control policies.

Ontologies that focus on SPECIAL AREAS OF SECURITY, are e.g., presented by Moyano et al. [138], who provide a conceptual framework for trust models that is also represented using UML. In 2010, Falkenberg started WS-Attacks[28], a Wiki that organizes web service attacks according to categories like the main security objective they violate (availability, integrity, confidentiality and access control), the number of involved parties, the web service component that is attacked, and he distinguishes attacks due to implementation weaknesses in web service frameworks and attacks due to conceptual flaws. Unfortunately, the Wiki is not kept up-to-date.

For mobile applications, an ontology is presented in [16]. Besides its focus on mobile applications, it includes assets, threats, vulnerabilities and mechanisms where mechanisms correspond to SecEval's notion of methods. For detailed evaluations of existing security ontologies, the interested reader is referred to [26, 202, 103].

In this thesis, we focus on vulnerabilities that are introduced into software during the requirements and design activities and do not go into much detail regarding vulnerabilities that are introduced according to PROGRAMMING LANGUAGE-SPECIFIC PARTICULARITIES (as can, e.g., be found for C, C++, Android, Java [126] and Perl in SEI CERT Coding Standards[29]). The reason is that errors in requirements and design are expected to be propagated into the implementation, which makes them expensive to correct. Nevertheless, we aim at avoiding overengineering due to loss of time in early phases of the development process and agree with Boehm and Basili that it is best to develop in a kind of "continuous prototype mode that still emphasizes getting things right early rather than late" [27]. As a consequence, we made sure that most parts of the work we present can be used for software pieces of various size, instead of following an all-or-nothing approach.

[27]Inheritance: If A and B are classes (concepts) and A has attribute x and B is a subclass of A, then B also has the attribute x.

[28]WS-Attacks. http://www.ws-attacks.org

[29]SEI CERT Coding Standards. https://www.securecoding.cert.org

2.3 Modeling Web Applications

In this section, we first discuss why graphical modeling can be advantageous, before introducing modeling approaches for web applications that take security into account.

2.3.1 Advantages of Graphical Software Modeling

Experienced practitioners frequently warn that a main challenge is to keep the need for security in mind while developing web applications, as functional requirements are more clearly visible when executing and testing the application. In addition, functional features can directly be invoiced, which can be difficult for security, as security is often taken for granted, both at the developer's and at the customer's side. All too often, a document containing security guidelines and company-wide security design decisions is handed over to the developers at the beginning of a project, but they are not in the focus until (mostly external) penetration testers point out vulnerabilities. In the OWASP community Munich, penetration testers reported, that they almost always find common vulnerabilities, according to the OWASP Top 10 and that in their experience, developers are often pressed for time when fixing reported vulnerabilities, as penetration tests are usually executed just before a release date. Therefore, several modeling approaches try to put the focus on requirements and design by connecting important requirements and decisions directly to the web application's architecture.

In general, we assume that well-presented graphical information leads to having a comprehensive view and a better understanding, which eases conscious decision making and therefore reduces common careless mistakes. To prove this assumption is beyond the scope of this thesis. However, it is common knowledge that engaging multiple senses is advantageous for understanding and learning. This is also reflected in the common phrase: "Do I have to paint (you) a picture?" [204] for something that seems not to be understood from reading or listening alone. Moreover, models that enrich textual documentation are thought not only to increase passive understanding, but also to foster discussions between practitioners (i.e., speaking and listening, which might lead to creating/changing models). This could similarly apply to software design models and to ontology representations. Nevertheless, graphical representations should be designed in a way that they are not contradictory or misleading, which can be found out by collecting experience using case studies that employ them.

2.3.2 Approaches that support Security-Related Requirements and Design Decisions

According to a survey, "86% of all websites had at least one serious vulnerability in 2012", which means that an "attacker could take control over all, or some part of the website, compromise user accounts on the system" or "access sensitive data" [94]. One way to counter this trend is to increase the awareness for security decisions, to make sure that necessary decisions are consciously taken during development and that these decisions have an impact on the actual implementation. Security-aware modeling approaches for (web) applications support this by visually or textually describing various properties of an application. Ideally, developers can quickly get aware of them and – for visual modeling approaches – may internalize main structures easily, which can serve as an anchor for further information they come across. In addition, many models make it possible to automatically generate artifacts like code snippets or configuration files. Fortunately, countless approaches for modeling secure applications exist; unfortunately, most of them are restricted to specifying access control policies, which is only one of many methods that have to be considered for building a "reasonably secure" application.

Note that there are risks for web applications in general as well as risks that should be rated individually for each use case (and it might turn out that some risks can be ignored). For modeling individual threats, we suggest modeling misuse cases, as presented by Sindre and Opdahl in [198] or mal-activities (activities by attackers who threaten a system and their impact on other activities), as presented by El-Attar in [77]. For detailed risk analysis and for modeling threat scenarios, the CORAS method [127] can be used. Although we assume that anticipated threats and risks are analyzed meanwhile or prior to modeling, we mainly focus on general risks for web applications, like those mentioned in the OWASP Top 10 [161]. Handling these common risks is necessary for implementing a "reasonably secure" web application.

Existing modeling approaches are briefly introduced in the following, starting with UWE, which we decided to further extend in this thesis. One of the cornerstones of the UML-based Web Engineering (UWE) [116, 44] language is the "separation of concerns" principle, which is implemented by using separate models for different views on the application. For example, the Content model contains the data structure used by the application, the Presentation model can be used to sketch the web application's user interface and the Navigation model is used to define which ways users can follow to browse through it. For each view, an appropriate type of UML diagram is used, e.g., class diagrams for the Content model. In addition, the UWE profile adds a set of stereotypes, tag definitions and constraints, which can be downloaded from the UWE website [224]. Already for her master's thesis [37], the author has chosen to start to extend UWE for modeling security features of web applications, primarily because it is flexible due to its views and can be modeled with any UML editor that supports profiles. Further details of the UWE modeling approach can be found in part III, in which we continue to enhance UWE so that it can be used to express a variety of security requirements and security-related design decisions.

UWE models can be built using any UML CASE tool that enables the use of UML profiles. We use the MagicUWE plugin, which was implemented by the author and others [40] as a MagicDraw [147] plugin. It provides additional support to the developer, e.g., repetitions can be avoided. Thus, instead of creating a basic element, as a class, and applying a stereotype to it, UWE's stereotyped elements can be inserted directly from the MagicDraw toolbar. Besides, several model transformations can be performed (semi-)automatically. Further functionalities of MagicUWE are described in part III, chapter 6.

In order to use transformation functionalities of MagicUWE in toolchains, we integrated it into a tool workbench, called Service Development Environment (SDE) [193]. The SDE has been developed within the SENSORIA project [194], an initiative funded by the EU from 2005 to 2010. Afterwards, it has been maintained and extended within the scope of the NESSoS and the ASCENS [7] EU projects. The SDE is an Integrated Development Environment (IDE) based on a service-oriented approach, where each tool is represented as a service. Technically, the OSGi framework in Eclipse [75] is used to integrate tools such that their functionalities can be connected and executed in a row.

ActionGUI [11] is an approach for generating complete, but simplified, data-centric web applications from models. It provides an OCL specification of all functionalities, so that navigation is only modeled implicitly by OCL constraints. In general, ActionGUI abstracts less from an implementation than UWE does, as its aim is to directly generate the entire code, which requires that all information has to be provided in the models. In the most recent version, which is presented by Basin et al. in [9], ActionGUI focuses on textual models.

UMLsec [109] is an extension of UML with emphasis on secure protocols. It is defined in form of a UML profile including stereotypes for concepts like authenticity, freshness, secrecy and integrity, role-based access control, guarded access, fair exchange, and secure information flow. In particular, the use of constraints gives criteria to evaluate the security aspects of a system design, by referring to a formal semantics of a simplified fragment of UML. UMLsec models, compared to UWE models, are extremely detailed and therefore quickly become very complex. Tool support is only partly adapted from UML1.4 to UML2 by some new tools [110].

SecureUML [125, 10] is a UML-based modeling language for secure systems. It provides modeling elements for role-based access control and the specification of authorization constraints. A SecureUML dialect has to be defined in order to connect a system design modeling language as, e.g., ComponentUML to the SecureUML metamodel, which is needed for the specification of all possible actions on the predefined resources. In [4], Alalfi et al. exemplarily reconstruct SecureUML rules from a web application written in PHP, by automatically creating UML sequence diagrams that present interaction behavior.

A similar approach as SecureUML is UACML [199] which also comes with a UML-based meta-metamodel for access control, which can be specialized into various metamodels for, e.g., Role-Based Access Control (RBAC) or Mandatory Access Control (MAC). Conversely to UWE, the resulting diagrams of SecureUML and UACML are quickly getting overloaded, as SecureUML uses UML association classes instead of simple dependencies and UACML does not introduce a dedicated model to specify user-role hierarchies.

Further approaches that model access control can be found in [33, 114, 137, 183, 146]. The interested reader is referred to semantic literature reviews, which were published in 2015 by Nguyen et al. [145] and van den Berghe et al. [219].

Rodríguez et al. add symbols to BPMN [178] and to UML [177] for modeling high-level security requirements, such as the need for access control, non-repudiation, integrity, privacy and points where attacks should be discovered and registered. Therefore, process flows are annotated by several symbols like a lock named `AC` for activities that should be placed under access control. The clear focus on requirements, without any refinement regarding design, let the diagrams remain well readable.

Other approaches address the modeling of security aspects of service-oriented architectures (SOAs), such as the SECTET framework [97], UML4SOA [88], and SecureSOA [133]. The SECTET framework proposes the use of sequence diagrams for the representation of a set of security patterns, in UML4SOA security features are modeled as non-functional properties using class diagrams, and SecureSOA relies on FMC block diagrams and BPMN notation. Hoisl and Sobernig [102] model confidentiality and integrity for SoaML invocation protocols.

Web engineering approaches that do not take security into account are, e.g., Web Application Extension (WAE) [61] (although Conallen included a short, chapter about general web security in his book), UWA [71], the flashWeb method [107], OOHRIA [132], OOWS [218], WebML [30] and IFML [152], which is based on WebML and focuses on user interface modeling. The interested reader can find an overview of non-security methods for web application modeling by Kappel et al. [111, table 3-1, p. 60] and more recent literature reviews by Schwinger et al. [191], by Valderas and Pelechano in [217] and by Aragón et al. [6].

Part II

Evaluating and Relating Security Concepts

Software engineers need to find effective methods, appropriate notations and tools that support the development of secure applications along the different phases of the Software Development Life Cycle (SDLC). Our evaluation approach, called SecEval, supports the search and comparison of these methods, notations and tools (cf. RQ1). In addition, SecEval can be used as a structure for ontologies or knowledge bases, which allows us to tackle RQ2.

This part is a revised and extended version of our papers published in [45, 46, 48]. The author would like to thank her co-authors of these papers, namely Martin Wirsing and Nora Koch for discussions and corrections.

Part II consists of two main chapters:

SecEval: A Framework for Evaluating Security Engineering Approaches. We elicit requirements and define the SecEval evaluation process in section 3.1. In section 3.2 SecEval's architecture is described, which comprises: (i) a Security Context model with assets, methods, tools, security properties, vulnerabilities and threats (knowledge objects) (ii) a Data Collection model that records how data is gathered when researchers or practitioners are looking for methods, notations and tools that solve a specific problem (iii) a Data Analysis model that specifies how analysis, using previously collected data, is performed.
For validating our approach, we interviewed international senior researchers, as we report in section 3.3. We like to thank the 14 (associated) partners of the EU project NESSoS, who contributed with their ideas to the improvement of SecEval.
Besides, SecEval is used for evaluating tools in the web testing domain. Under the supervision of the author, Stephanie Schreiner [188] contributed a comparison of security-related tools and methods on a high level of abstraction and Christian Lacek [122] executed an in-depth comparison of web testing tools with practical tests. These bachelor's theses served as the data basis for our SecEval models in section 3.4.

SecWAO: A Secure Web Applications' Ontology. Regarding the area of web applications, a reason for vulnerabilities is that for developers the term "security" is difficult to grasp. Many security properties exist and there are various methods to enforce them and methods to prevent common vulnerabilities in web applications. Ontologies can help to get a comprehensive view of web security and to structure this domain by relating relevant knowledge objects. Our ontology, called SecWAO, is based on SecEval and supports teaching purposes and web developers when specifying security requirements or making design decisions. Besides, it serves as a basis for our security extension of the UWE notation, which will be presented in part III.
In section 4.1, we introduce SecWAO by example, while mapping concepts from the area of Cross-Site Scripting (XSS) to knowledge objects of SecEval. SecWAO is then presented with excerpts of its security properties (section 4.2), methods (section 4.3), vulnerabilities and threats (section 4.4).
On the basis of the author's ideas and requirements, published in [45]), Martin Reithmayer [176] developed a prototype of an online knowledge base (section 4.5.2). Beyond others, it allows users to create, edit, relate, and search for knowledge objects of SecEval's Security Context model. Knowledge objects can stem from a SecEval evaluation as well as from SecWAO.

The interested reader can download[30] all models in the MagicDraw [147] or XMI [150] format.

[30]SecEval, its extensions, case studies, and SecWAO. http://www.pst.ifi.lmu.de/~busch/SecEval/

Chapter 3

SecEval: A Conceptual Framework for Evaluating (Security) Engineering Approaches

In this chapter, we describe SecEval, our conceptual evaluation framework for methods, notations and tools, supporting the development of secure software and systems. Our framework supports collecting security-related data and describing security-relevant metrics, using them for reasoning and obtaining the appropriate techniques for a specific project. An example for a simple evaluation is required to answer the question posted in the implementation phase: “Which library for authentication should be used?” A more elaborated one could be the evaluation of risks for a concrete software system.

The conceptual framework comprises a structural part and a behavioral part, defined as a model for evaluation and an evaluation process. For the graphical representation of the evaluation model a UML class diagram was chosen; the evaluation process is represented as a UML activity diagram.

This chapter is a revised and extended version of our papers published in [45, 46]. We present the requirements engineering work that was done to elicit the main steps of the process in section 3.1, followed by the architecture of SecEval in section 3.2. A guided interview (section 3.3) and a case study (section 3.4) validate the soundness of the SecEval approach.

3.1 Evaluation Process

We start by eliciting the requirements of such a framework, i.e., which stakeholders are involved, which concepts play a role in secure software and evaluation of methods, tools and notations, and how these concepts are related (cf. section 1.2, RQ1). Therefore, the first step was to name common stakeholders for secure software: security engineers (i.e. security designers and developers), normal users of the software and attackers. In some cases, users can attack software without being aware of it, e.g., when they have a virus installed on their computer. We consider those users also attackers, as well as developers who are, e.g., trying to smuggle malicious code into software. Figure 3.1 depicts stakeholders and use cases in a UML use case diagram.

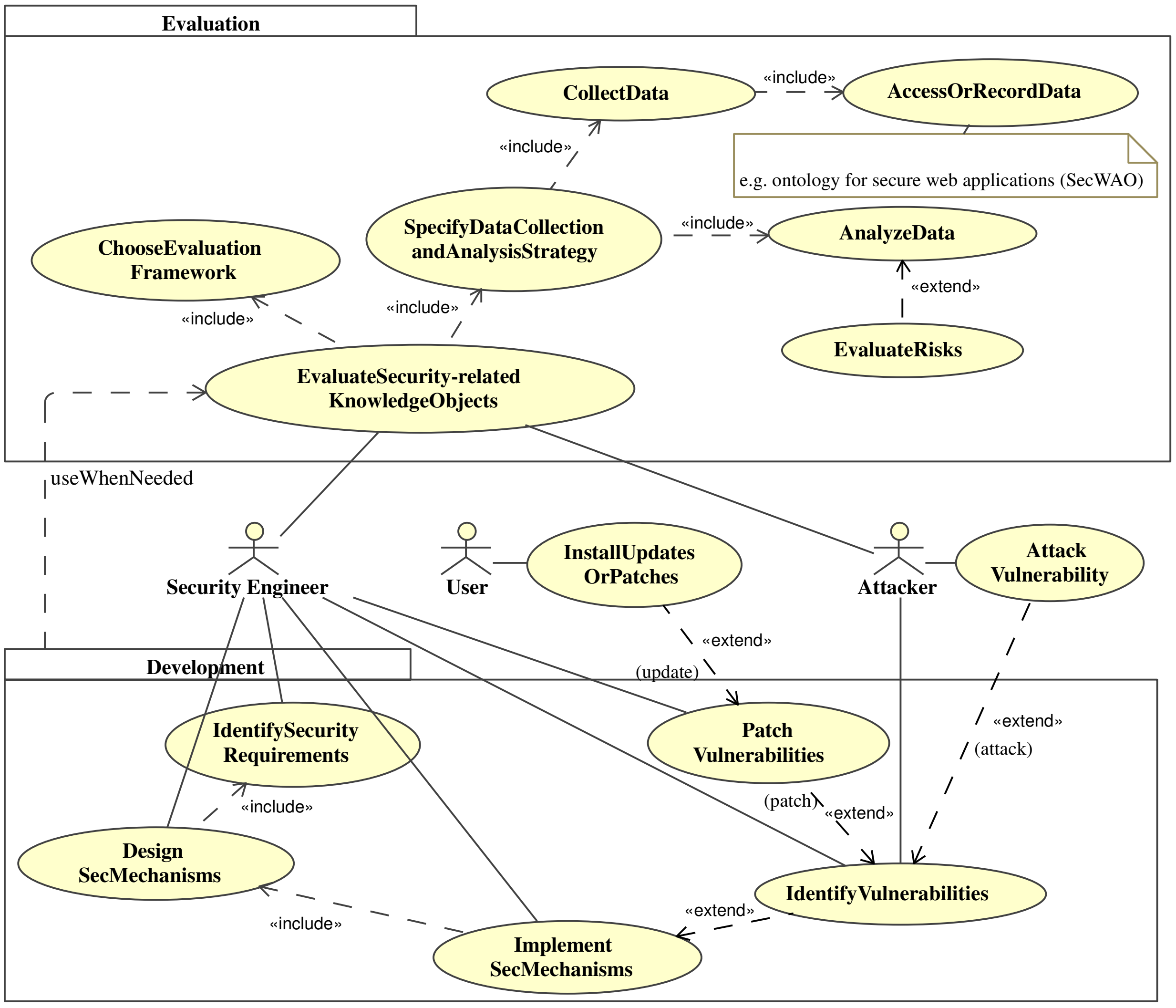


Figure 3.1: SecEval: stakeholders and use cases

We grouped use cases based on their purpose in evaluation and development use cases. The `Evaluation` package at the top contains all use cases related to collecting, reasoning and selecting, e.g., tools, whereas the package `Development` at the bottom of the diagram refers to security-related tasks within the SDLC, such as identification of security requirements, design and implementation of these security requirements, identification and patch of vulnerabilities. The «include» dependencies show the order these use cases have in the SDLC[1]: implementing secure software requires having a design, and a design implies that requirements were elicited beforehand. Both, the attacker and the security engineer can identify vulnerabilities, whereas the former usually attacks them and the latter tries to patch them, which is modeled using an «extend» dependency. Those patches can then be installed by users (which might be initiated by an automatic update function).

From time to time, tasks within the development package require evaluation activities to respond for example to questions like "Which tool should be used for gathering security requirements or for designing secure web applications?". In fact, for security experts, it is helpful to be aware of common security methods and tools that can be used for a specific task.

A generic evaluation process with SecEval consists of data collection and data analysis, as depicted in figure 3.2. The first step of the process is the data collection based on the current research questions. Therefore, different sources (as papers, reports, websites, ...) are gathered, which are then analyzed in the second step. This analysis process consists of extracting information from the data collected, activating some reasoning activities and expressing the results using SecEval's Security Context model. Notice that this process has to be adapted (and usually simplified) for a specific evaluation. Writing down the exact process might not always be necessary, as many tasks can be executed in parallel or in any order, which is indicated by horizontal bars.

In practice the basic ingredients of the evaluation process are a set of tasks that have to be performed and information pieces relevant for these tasks. Tasks, as e.g., selecting queries, searching or executing experiments, and defining filters, are represented as UML activities. Information pieces are represented as objects in the UML model, denoting the required input and provided output of a task. Examples for identified objects are: research questions, used resources, queries, filters and criteria, which will be detailed in the next sections.

3.2 Architecture

The use cases from our requirements analysis and the objects of the evaluation process are a starting point to identify relevant concepts related to security for using and evaluating methods, notations and tools during the software engineering process. We cluster these concepts in three packages, which are presented in this section: Security Context (section 3.2.1), Data Collection (section 3.2.2) and Data Analysis (section 3.2.3). Section 3.2.4 concludes with two extensions of SecEval.

Figure 3.3 gives an overview of SecEval represented as a UML class diagram[2] that can be instantiated with concrete methods, tools and notations whenever needed. As introduced in

[1]Please note that our approach is *not* based on a particular software development process (be it waterfall, iterative or agile), as we refer to tasks within phases of a Software Development Life Cycle and do not describe whether this "circle" is cut open, rather big or just concerning a single task.

[2]The background of classes that can directly be instantiated is colored and the names of them are not italic. The same colors are used in following figures, which introduce each model in more detail.

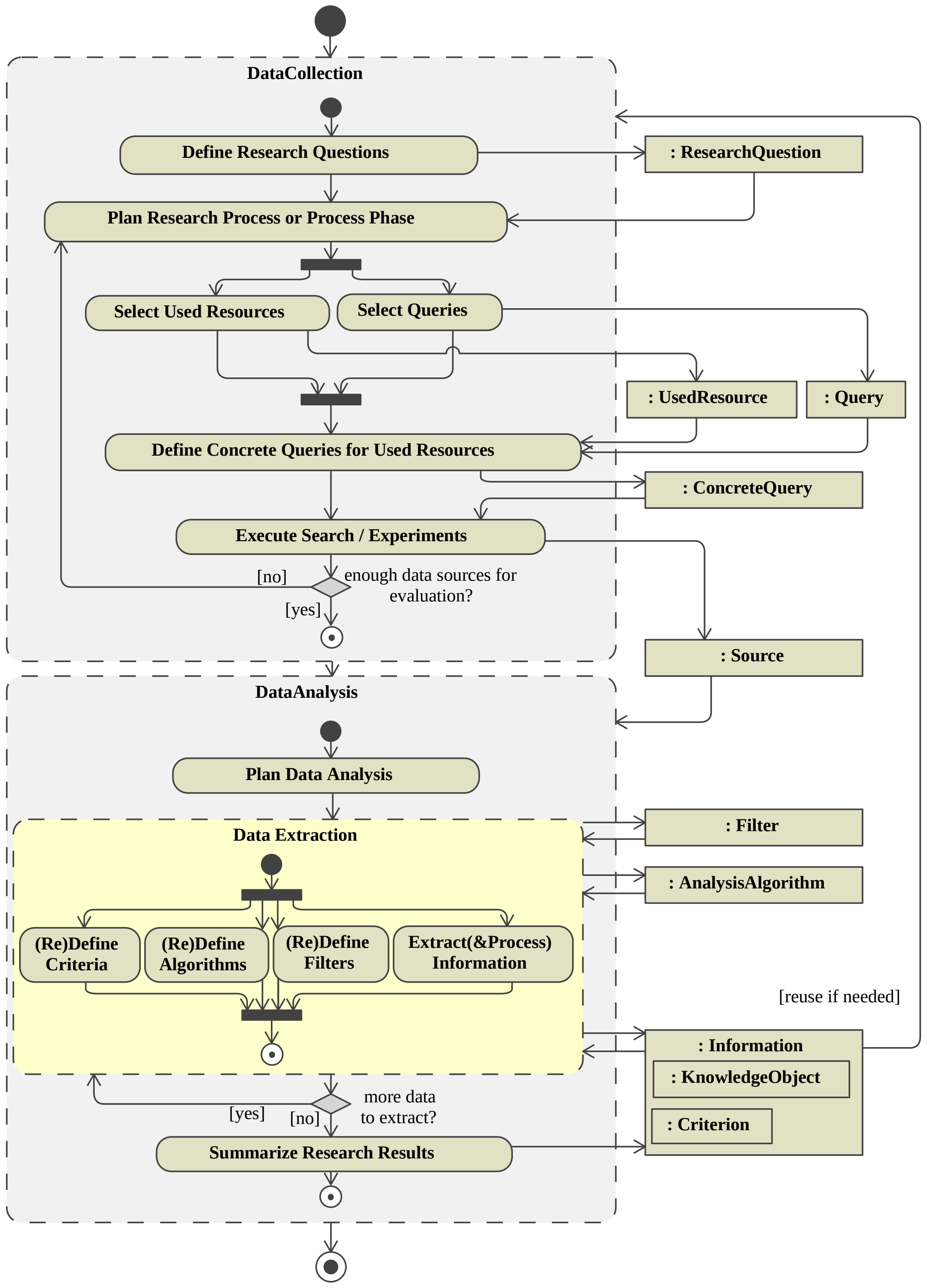


Figure 3.2: SecEval's evaluation process

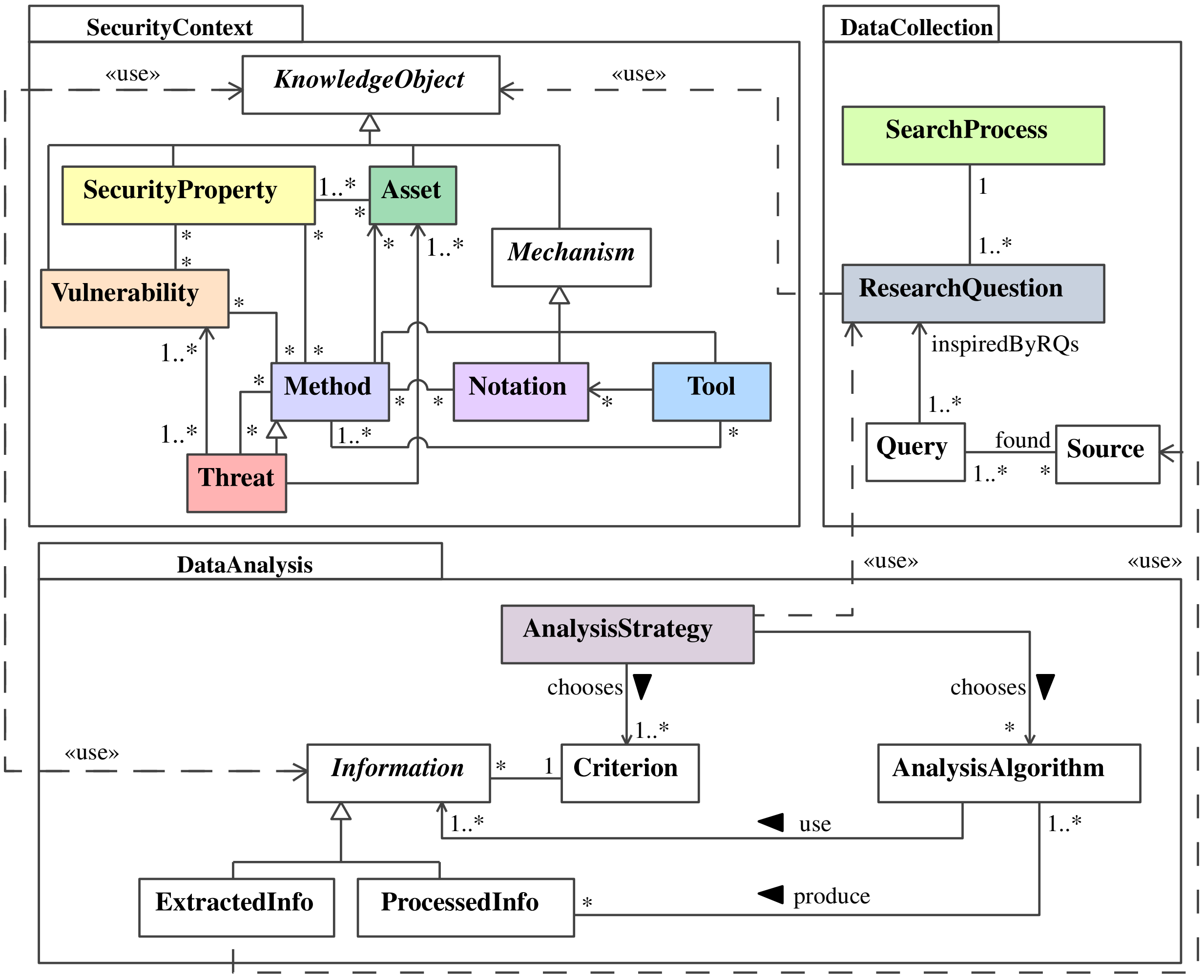


Figure 3.3: SecEval: model overview

section 2.2.2 on page 17, we consider the following concepts as key elements in SecEval: assets, methods, notations, tools, security properties, vulnerabilities and threats.

3.2.1 Security Context

SecEval's Security Context model provides a structure for the classification of methods, notations and tools together with assets, and security properties, vulnerabilities and threats. Within this model we represent a security feature as a UML class, called `SecurityProperty`, and an abstract class called `Mechanism` is used from which the classes `Method`, `Notation` and `Tool` inherit common attributes such as `goals` or `costs`. We focus on security aspects, but the model can also record non-security mechanisms.

In figure 3.4, enumerations are edged in grey and attributes and roles are typed; however, the types are not shown in the figures due to brevity. The main characteristics are specified as Boolean types (can.., has.., is..). In an implementation of our model, it should be possible to add further items to the enumerations.

We adopted the abstract class KnowledgeObject, which is used in the Common Body of Knowledge (CBK) [49], as a super class for all context elements that are described by SecEval. In SecEval, we applied separation of concerns so that only very general descriptions remain as attributes in a knowledge object, which can be applied to all elements (cf. figure 3.4). The class `KnowledgeObject` has associated names, tags and related sources, which could be any kinds of sources, as publications or Uniform Resource Locators (URLs). For connecting and describing ambiguous knowledge objects that are not (yet) clearly separated in practice, we created the UML association class `Ambiguity`. For convenience, we allow to group knowledge objects in categories, which themselves can belong to knowledge objects. Creating categories is especially useful if instance diagrams get bigger.

Security properties, such as confidentiality[3] or integrity[4] are represented by the class Security Property. Security properties can also be based on other security properties.

Security properties are always related to Assets. For us, an asset is something of value that has to be protected [200, p. 303], as web servers, web applications or information like passwords. For example, the security property "confidentiality" can be a feature of a transmitted data package. Assets can belong to or be used by other assets and some of them, as web servers, might be described by a certain configuration. In addition, an asset can include `Mechanisms`. For example, authentication is a `Method` and thus a `Mechanism` and an authentication system might be an asset that includes the concept of authentication.

A Vulnerability is "a weakness that makes it possible for a threat[5] to occur" [25, p.498]. Thus, it endangers security properties. Examples are injection[6], buffer overflows, etc. The objective of certain methods is to detect vulnerabilities or to shield them from being exploited by a threat. A vulnerability can have a location; details about the component the vulnerability is located in can be given using the attribute `affectedComponents`. In case the method, notation

[3]Confidentiality "is the concealment of information or resources" [25, p.4].

[4](Data) integrity "refers to the trustworthiness of data or resources, and it is usually phrased in terms of preventing improper or unauthorized change" [25, p.5].

[5]"A threat is a potential for a security breach of an asset" [189].

[6]"Injection flaws occur when untrusted user data are sent to the web application as part of a command or query" [171, p.9].

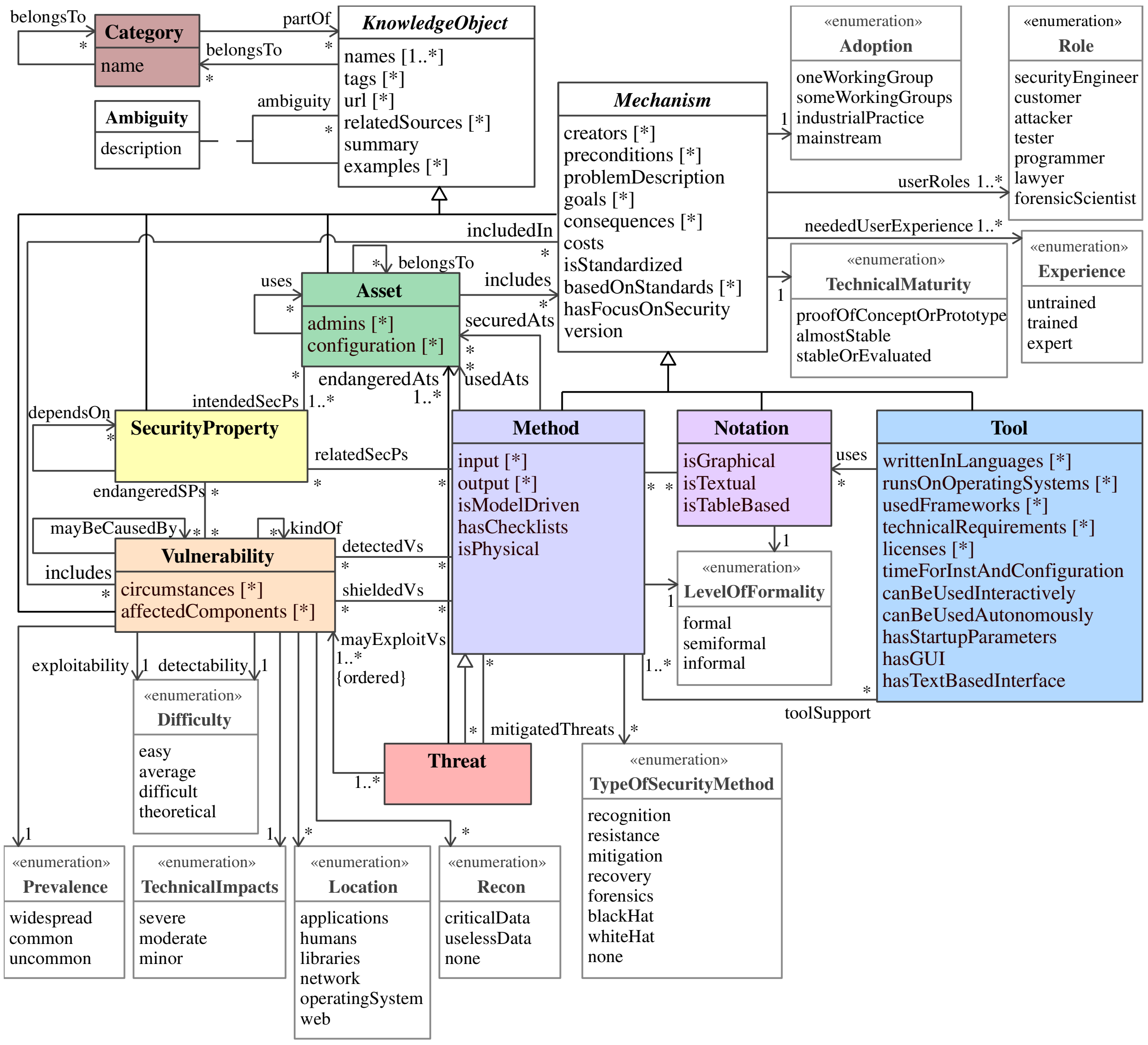


Figure 3.4: SecEval: security context

or tool that contains the vulnerability is already modeled with SecEval's Security Context model, the role `includedIn` can be used. Looked at the difference between the terms "weakness" (used by CWE [66]) and "vulnerability" the other way around than we did in section 2.2.2, a vulnerability is a kind of weakness that is implemented in a piece of software. In SecEval, we do not differentiate between both terms, as a `Vulnerability` that has not just been introduced due to a technical issue while programming, is naturally more general. For example, conceptual vulnerabilities in methods (cf. `includedIn`) affect all available implementations of this method.

Furthermore, the categorization scheme from OWASP Top 10 [161] is included (which is adapted from the OWASP Risk Rating Methodology [166]) using prevalence, impact level, detectability and exploitability. Regarding the latter two roles, the `Difficulty` "theoretical" means that it is practically impossible to detect or exploit a vulnerability (cf. figure 3.4). For concrete vulnerabilities, the inherited attribute `name` can refer to an identifier of the Common Vulnerabilities and Exposures (CVE) database [213]. SecEval uses the enumeration `Recon` to describe possible leakage of (critical) data.

In his "Broad Street Taxonomy" [197, p.394], Shostack differentiates between "Socially Engineered Vulnerabilities" (the user is tricked and does not intend to run software), "User-Interaction Vulnerabilities" (require user interaction, but no deception) and "Classic Vulnerabilities" (do not require user interaction). We decided not to add this distinction, as the terms can easily be misunderstood. If needed, we suggest to directly model the decisions as Boolean attributes, instead. For example, a `Vulnerability` could have attributes like `needsUserInteraction` or `needsDeception`.

A Threat is "a potential violation of security" [25, p.6]. It is treated as a kind of method that is vicious. (Note that we do not define an "attack" here, because general SecEval models should be time-independent – attacks threaten assets at a certain time – and independent of concrete assets.) At least one vulnerability has to be involved, otherwise a threat is not malicious (and the other way around), which is denoted by the multiplicity [1..*]. Additionally, threats can be mitigated by other methods. Mitigation is not necessarily partial; it can also stand for resistance. Threats endanger at least one asset and general methods can secure assets or use them. E.g., cryptographic hashing secures passwords by not storing them directly and authentication uses passwords to identify users.

A Mechanism is described by a problem statement, by the goals it strives for, by its costs and by the consequences it implies. Mechanisms can be based on standards or be standardized themselves. Before applying a mechanism, the preconditions that are necessary for using it have to be fulfilled. Furthermore, an estimation regarding technical maturity and adoption in practice might be given. It can also be expressed whether or not the mechanism has a special focus on security, because in practice many mechanisms might also be used for security purposes, but do not directly focus on them. Several levels of usability can be stated indicating the experience users need in order to employ a mechanism.

The classes Method, Tool and Notation inherit all these properties from the abstract class `Mechanism` and have their own characteristics defined by a set of specific attributes. For example, a Method has some general attributes, such as input, output and if it is model-driven or provides checklists for developers. A method can be supported by notations or tools; this is represented in the model with corresponding associations between the classes. According to [14, fig.9.3], a method can support different security tactics, which can be denoted by elements of the enumeration `TypeOfSecurityMethod`. It can detect attacks (recognition), resist attacks (resistance), react to attacks (mitigation) or recover from attacks (recovery). In addition, it can be used for forensic tasks and sometimes it is interesting whether a tool is mainly used by white hats, or black hats. For a NOTATION, we consider characteristics such as whether the notation is graphical, textual or based on a tabular representation. We also added a level of formality, which ranges from informal to formal. The description of a TOOL is given among others by the information of languages it is written in, operating systems it supports, frameworks it uses and licenses under which it is released. The needed time for installing and configuring can be provided. Booleans describe if the tool can be used interactively or autonomously, if it has start parameters, a GUI or a text-based user interface.

Details of Tools and Methods according to the SDLC

During our experience with the CBK in the NESSoS project, we noticed that tools as well as methods would better be described according to the phases of the SDLC, because attributes that are used to describe a method or tool are related to the SDLC phases they cover. We considered the following phases for the SDLC: requirements, design, implementation, testing, assurance, risk & cost management, service composition, deployment and runtime. They are partially based on the phases of the development process defined in NESSoS [41]. In our experience, no phase-related attributes are needed to describe features of notations.

We enrich the class `Method`, as depicted in figure 3.5, by delegating the description to the abstract class `MAreasOfDev`, which is a wildcard for detailed information about the method according to the aforementioned phases. For example, a method as Microsoft's Security Development Lifecycle [124], can be used as a basis for designing secure applications, but also covers other phases. In this case, the attributes of the classes `DesignM` and `ImplementationM` and others would be used to describe this method. For SecEval, we selected appropriate attributes according to what we missed when describing tools in the CBK, as e.g., the level of detail for methods that are applied during requirements phases and which tasks they comprise (elicitation, analysis, specification or management). For testing, we consider the area that is tested (web, network or system) and how many information the tester got (whitebox, blackbox or greybox test[7]). Methods that are applied during the runtime of developed applications can be classified by the languages the system has to be written in order to apply the method and by the influence a method can have on the running application, as altering or stealing data or inspecting data flow. The meaning of all depicted attributes is detailed in appendix A.1.

Similarly, figure 3.6 depicts our `Tool` class and the abstract class `TAreasOfDev`, which is a wildcard for detailed information about the tool in relationship to the phases of the SDLC. A tool can be used in several development phases and as it supports at least one method, it is indirectly described by method attributes so that only tool-specific attributes had to be added to descendants of `TAreasOfDev`. We decided to use abstract attributes for methods and technical-related ones for tools. For example, for a requirements tool, we use a Boolean to document if it can import handwritten sketches. Likewise, it might vary from tool to tool whether an assurance tool can export a proof using LaTeX, even if they support the same assurance-related method. Regarding deployment, we consider attributes for documenting how the described tool technically interacts with the software that is to be deployed. E.g., a deployment tool might be

[7]For a whitebox test, the tester knows how the application under test is implemented. For a blackbox test, the tester just observes input and output. Greybox testing is a mixture of both.

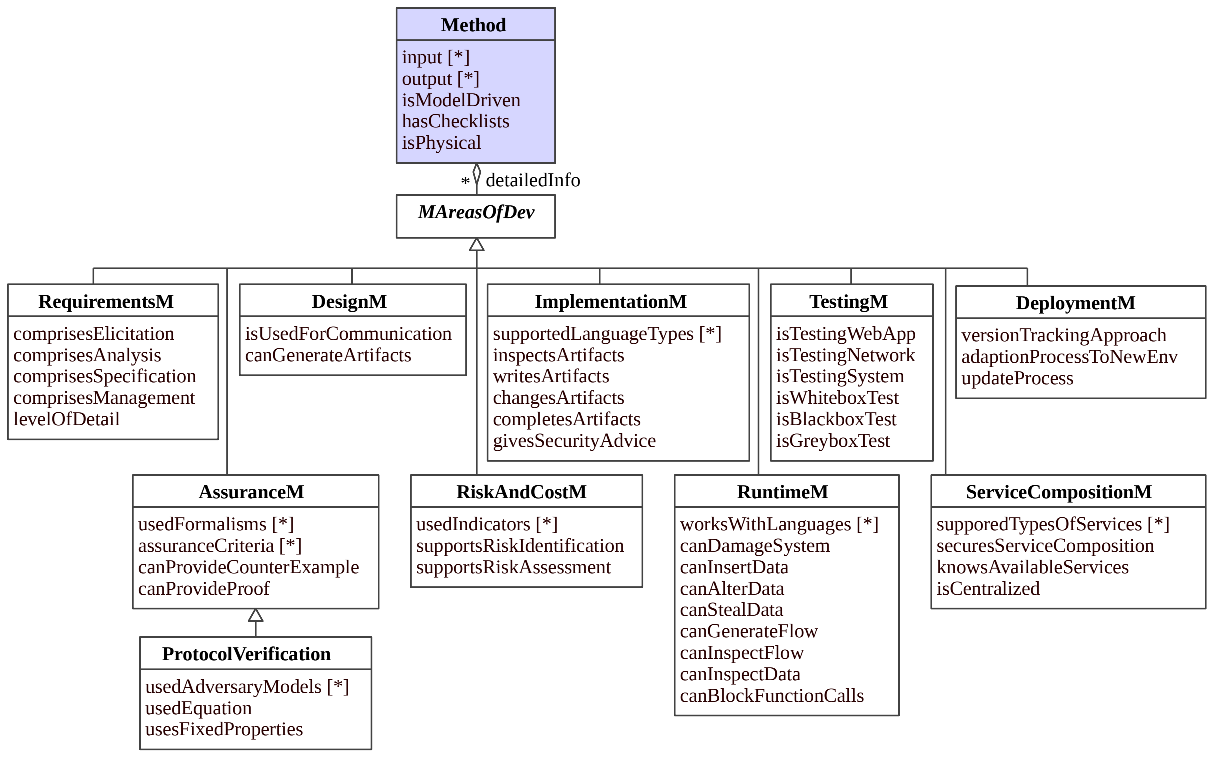


Figure 3.5: SecEval's Security Context model: details of methods

able to install, configure, update or uninstall software. The meaning of all depicted attributes is detailed in appendix A.2.

Figure 3.6 also depicts how tool-chains are built: `Functionality` can be grouped to sets that belong to tools. For us, a functionality is a tool or a part of a tool that covers a certain feature. It might be described by its typed input or output and it can be used to build toolchains. In a toolchain, output from predecessors is used as input for successors.

Relations between Methods, Notations and Tools

As seen before, a tool supports a certain method. However, we have not yet defined the quality of this support. Does the tool fully support the method? Does it provide partial support? Which features are not supported? We add this information to the model using the association class `ToolSupportedMethod`, as depicted in figure 3.7. The association class itself is inherited from the class `Method`, thus can redefine its attributes. For instance, a design tool can partly support a model-driven method (e.g., by facilitating the modeling process), although it cannot generate artifacts. In this case, `DesignM.canGenerateArtifacts` (cf. figure 3.5) would be set to `false`.

Notations can be based on other notations, e.g., many context-specific extensions for UML exist that are also based on UML itself. Figure 3.7 depicts that a notation can belong to another notation. For example, Javadoc belongs to Java [159] and extends it, whereas the Scala [155] programming language is just inspired by Java (among others).

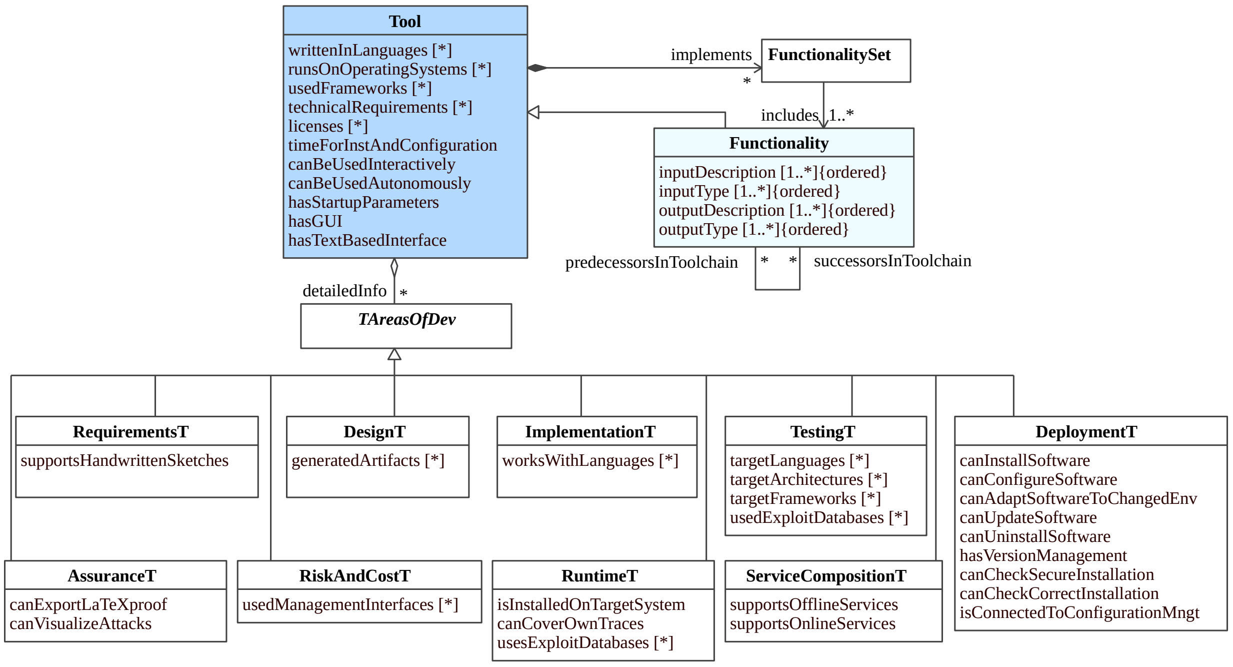


Figure 3.6: SecEval's Security Context model: details of tools

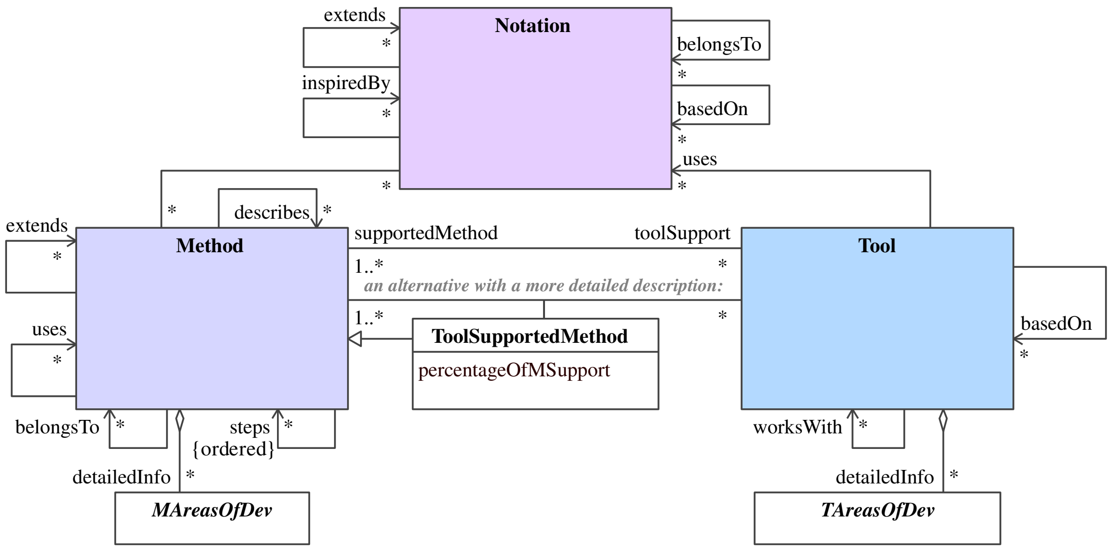


Figure 3.7: SecEval's Security Context model: further associations

A METHOD can extend other methods, which means it might also change them. In this case the role `extends` should be further specified, we recommend adding an association class that inherits attributes from the class `Method` (similar to the association between method and tool). In this way, it can be exactly described if and how the original methods are modified. It is also possible that other methods are used without any changes (role `uses`) or that methods belong to other methods (e.g., "web security testing" belongs to "security testing"). A method can also describe another method or it can define an ordered list of steps it comprises.

A TOOL can be based on other tools, which is the case when libraries are used or when plugins are written. The association `worksWith` denotes that tools work together, as e.g., in a toolchain.

In general, many ways exist to model certain facts. For example, a whitebox test can be modeled as an instance of the class `Method` and other method instances can use it (denoted by the association `uses`) to describe that they are testing methods. However, for famous methods it can be handy just to set a Boolean, like `TestingM.isWhiteboxTest` to express the same fact.

We encourage users of SecEval, to adapt the Security Context model dynamically, so that for a method m that is heavily used, an attribute called "`is`" + `m.names.head` is added to a descendant of `MAreasOfDev` that fits best. Another approach is to scan famous sources as Wikipedia from time to time to parse listings as "software testing types"[8] in order to add its headlines to an enumeration named `TestingTypes`, which can then be connected to `TestingM` with an unidirectional association. This would result in the possibility to directly describe a method m that uses acceptance testing, by setting `m.detailedInfo.testingTypes.isAcceptanceTesting` to `true`. For us, the reason for not adding that many specialized attributes at the first step was twofold: First, we wanted to keep our models clean and second, our main focus is on uncovering and depicting dependencies between knowledge objects using associations (cf. chapter 4), which would be undermined by using surrogate attributes in the long run.

3.2.2 Data Collection

High-quality data is the required basis for obtaining good evaluation results. Therefore, we create a rigorous schema which describes a set of properties that have to be defined before starting collecting data. The Data Collection model we build contains all relevant features that are needed during data collection. It is based on Kitchenham's systematic literature review [113]. Conversely to Kitchenham's approach, we do not restrict ourselves to reviewing literature; we also include, e.g., information about tools which cannot always be found in papers, but on websites and in results from benchmarks or experiments.

SecEval's Data Collection model is depicted in figure 3.8 and closely relates to the process of data collection and data analysis depicted in figure 3.2. Similar to Kitchenham's literature review, RESEARCH QUESTIONS are used to define the corner stones and the goals of the search, i.e., define what is inside and outside the scope of research. Before specifying the research questions, a basic understanding of the context is needed, therefore a dependency, stereotyped «use», points from `ResearchQuestion` to the package `SecurityContext` in figure 3.4.

QUERIES are used to find matching sources containing data which might help to answer research questions. As different search engines support different types of queries, CONCRETE

[8]Wikipedia: Software Testing Types. https://en.wikipedia.org/wiki/Software_testing#Testing_Types

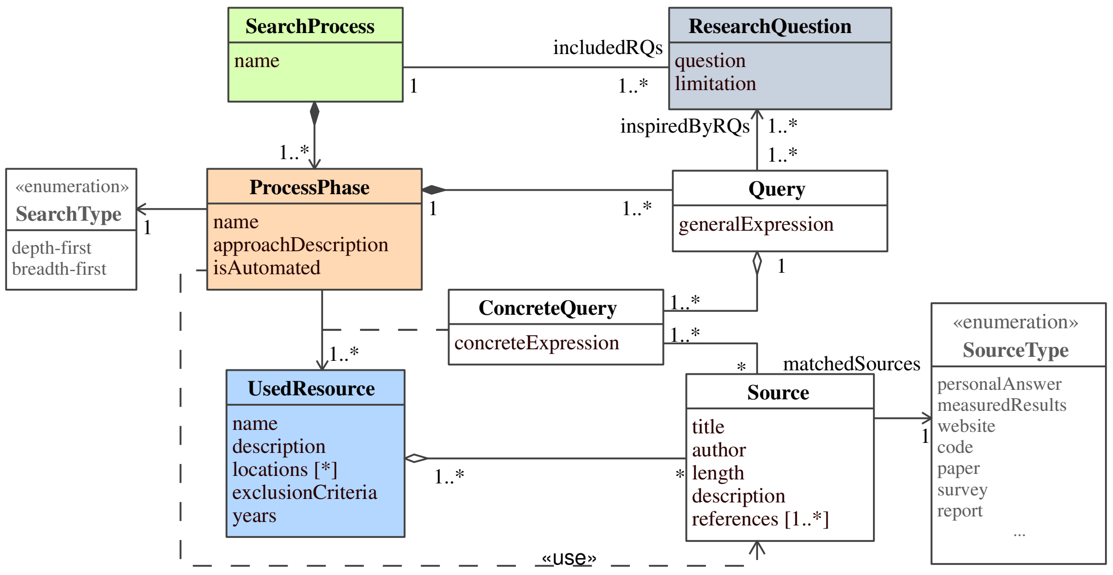


Figure 3.8: SecEval: Data Collection model

QUERIES are specific for each resource. In figure 3.8 the use of the association class `ConcreteQuery` (depicted by a dashed line) denotes that for each pair of `ProcessPhase` and `UsedResource`, the class `ConcreteQuery` is instantiated. The concrete search expression is derived from a general search expression. Thus, concrete queries document what was searched for in a certain resource so that results remain as repeatable and comprehensible as possible. At the same time, concrete queries belong to a general query. Queries can also refer to questions that are used as a basis for experiments (cf. case study in section 3.4).

For example, the general search expression could be "recent approaches in Security Engineering" and we want to ask the Google Scholar[9] and a popular researcher. Consequently, both are our USED RESOURCES. For Google Scholar we could use "`"Security Engineering"` 2014..2015" as a concrete search expression and the concrete expression for asking a researcher could read: "I am interested in Security Engineering. Which recent approaches in Security Engineering do you know?". It is worth to notice that resources can not only be scientific papers but also the web or measured data, code repositories or persons. When measuring data, the resource's exclusion criteria have to be empty or specified in a way that the result is not distorted.

If a concrete query matches SOURCES, as papers, websites or personal answers, we classify the source at least by author and description and provide information about the type of source and at least one reference in which resource to find it.

The SEARCH PROCESS can be performed in different ways, e.g., a PROCESS PHASE can be automated or not, or it can be a depth-first or a breadth-first search (cf. `SearchType` in figure 3.8). Depth-first means that the aim of a search is to extract a detail information about a relatively small topic, whereas a breadth-first search is good to get an overview of a broader topic. We noticed that Kitchenham et al. do not use more than one search phase. This might be the case because they prefer to not be biased by findings from a previous phase. The advantage of more than one search phase is that resources can be added which were considered interest-

[9]Google Scholar. http://scholar.google.com

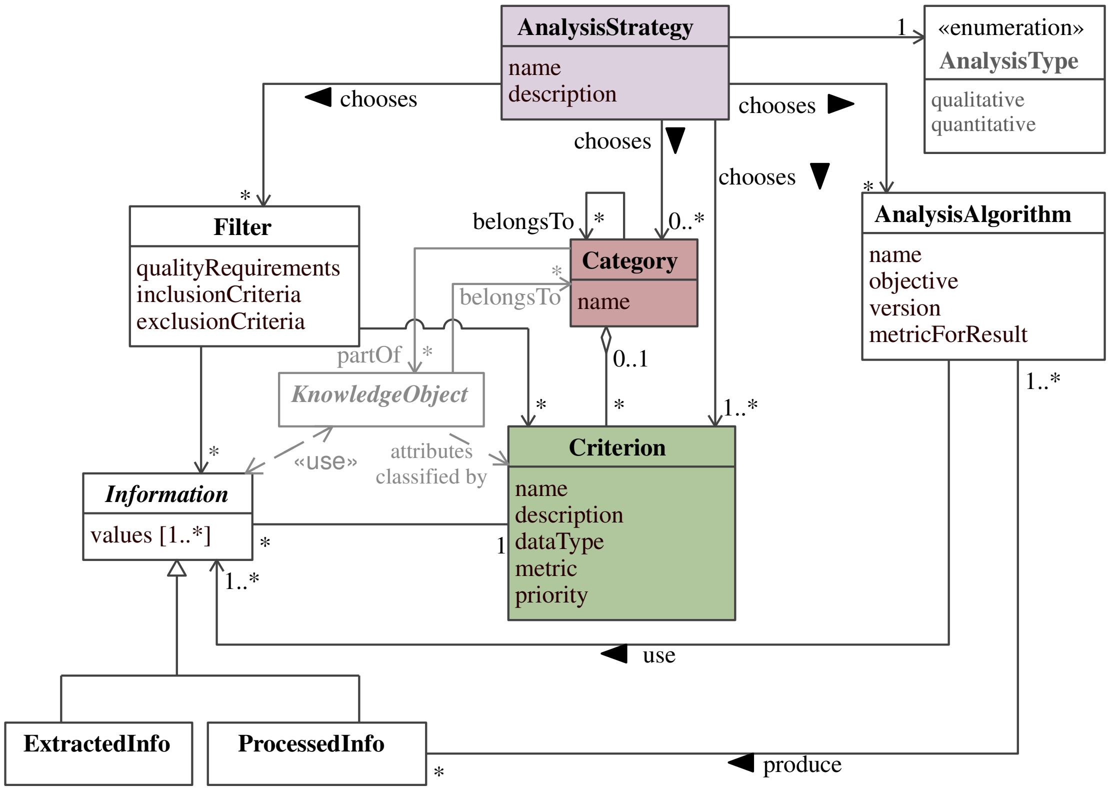


Figure 3.9: SecEval: Data Analysis model

ing in a previous phase, such as further contributions from conferences or from authors which seemed promising. Searching in several phases is especially useful for breath-first searches that are followed by depth-first search phases.

3.2.3 Data Analysis

Data is collected with the purpose to obtain an answer to research questions based on the analysis of the data. Figure 3.9 depicts our Data Analysis model, which relates relevant concepts for analyzing data. First, we have to specify which type of ANALYSIS STRATEGY we want to use. Are we limited to quantitative analysis or do we focus on qualitative analysis? Accordingly, one can later refer to Kitchenham's checklists for quantitative and qualitative studies [113, tables 5 and 6] to ensure the quality of the own answers to research questions. The analysis strategy requires selecting the used algorithms for analysis, categories & criteria, and filters according to the research question.

The chosen ALGORITHM might be implemented by a tool, but generally does not have to be executable on a computer. For example, the relation named `IsCompatible_NxN_ToolIO` can be seen as instance of an analysis algorithm. It expresses that "two notations are compatible if there exists a toolchain that can transform the first given notation into the second one" (as formally defined in [41]). In this case, the algorithm might contain a (semi-)automated depth-first search for a tool-chain consisting of tools where the output of one tool serves as input for the second one, etc.

Information can be extracted from the sources that were found in the data collection phase (cf. ≪use≫ dependency starting from the class `ExtractedInfo` in figure 3.3), or they can be processed using an analysis algorithm. Information can be stored in an instance of our Security Context model, which provides a sound basis for collecting and relating data about all kinds of knowledge objects. Additionally, the information classes can be used to describe information that is not directly related to an instance of a knowledge object or not meaningful without its connection to a concrete analysis process.

A criterion gives more information about data values, as e.g., attributes of knowledge objects, as it defines the data type (string, list of Booleans, ..) and the metric (milliseconds, ..). In addition, a priority can be defined, which is useful when methods, tools or notations should be compared by given criteria. Whether or not a criterion is grouped by categories depends on the kind of information which should be associated to the criterion: information with a single value does not have to be grouped, whereas it makes sense to group a set of values that are closely connected in terms of content. Another purpose of categories is to group knowledge objects, as described in section 3.2.1.

Besides, a filter can be specified to disqualify results according to certain criteria as costs or quality. This filter is finer grained than the filter that is defined by `UsedResource`'s attribute `exclusionCriteria` used in the Data Collection model, which only can be based on obvious criteria, such as the language the source is written in. In addition to this, the filter for data analysis can be based on extracted or processed information as well as criteria and thus can exclude, e.g., information about methods, tools or notations from the evaluation that do not meet a high-priority requirement.

3.2.4 Architecture Extensions

As the core of SecEval cannot include all attributes that could possibly be needed in the future, SecEval's models are extensible, which means that users can add classes and attributes for their domain of research. In this section, we introduce an extension to show how SecEval's Security Context model can be enhanced in order to support OWASP's Risk Rating Methodology [166]. In addition, we provide an extension for Moody's method evaluation approach [136].

However, we would refrain from expressing specialized data structures with SecEval., as e.g., those presented by Akhawe et al. [3] to formalize the http protocol with its entities, tokens, its status and other items that are closely interconnected by many different types of relations. Although the UML class model is capable of expressing this structure, it probably provides too many details in the context of SecEval.

OWASP's Risk Rating Methodology

To rate risks for running IT systems is a common task for security engineers. OWASP's Risk Rating Methodology provides categories and terms for this task. Figure 3.10 depicts the extended model whereby added connections use thick connection lines.

The class `Threat`, known from the basic Security Context model, inherits its features to a concrete `Attack`. The severity of the risk (which is an attribute of `Threat`) can be calculated by likelihood multiplied with impact. The likelihood is derived from the factors which describe

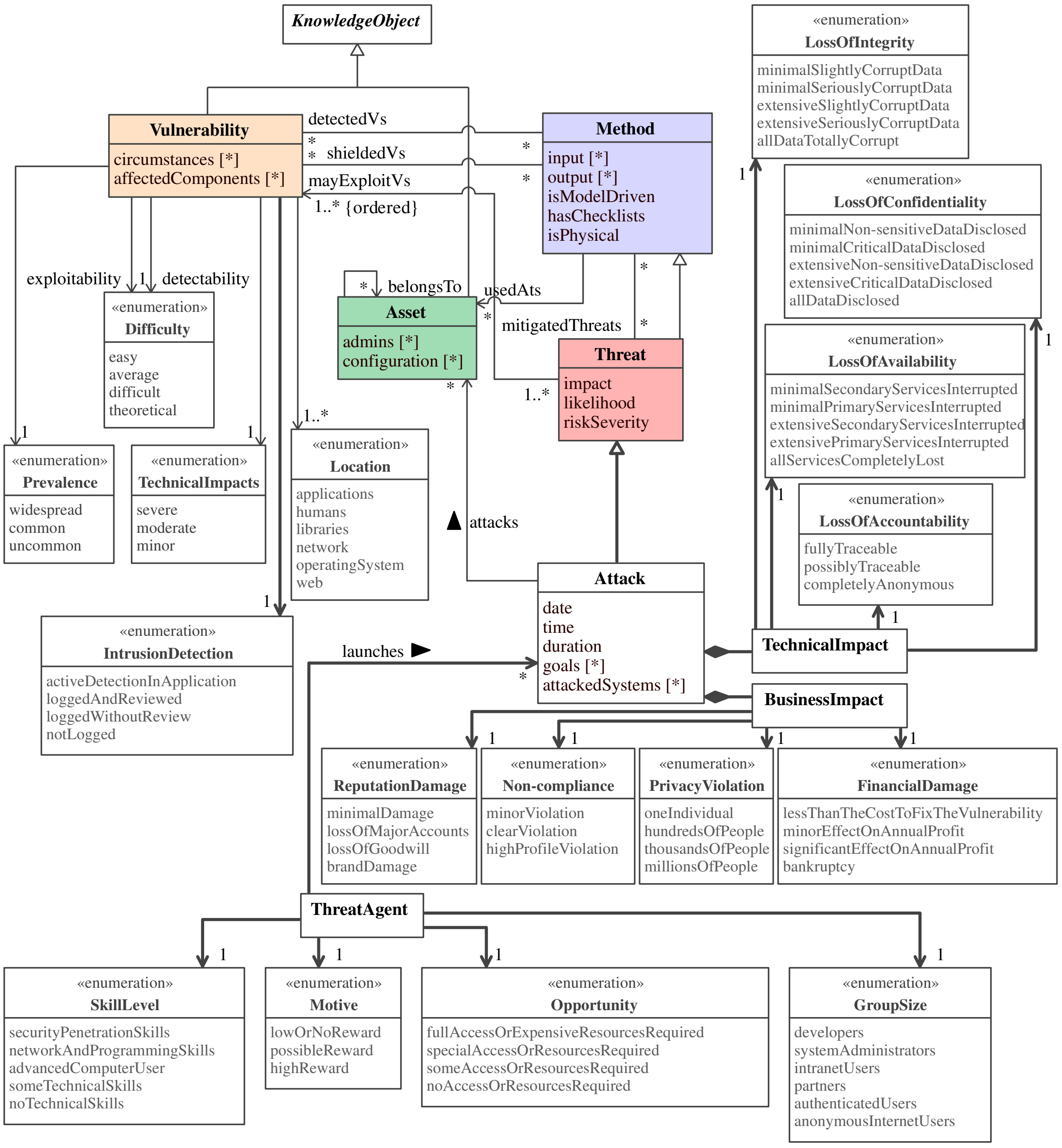


Figure 3.10: Inclusion of basic risk evaluation approach (Security Context model)

the vulnerabilities and the threat agents, whereas the impact is determined by the concrete technical and business-related consequences. Therefore, each enumeration's literal is mapped to a likelihood rating from 0 to 9. For example, OWASP ranks the motives as follows: "Low or no reward (1), possible reward (4), high reward (9)". For more information, the interested reader is referred to [166].

Alternative risk rating approaches are, e.g., NIST's Common Vulnerability Scoring System (CVSS)[10] or Microsoft's simple DREAD model[11].

Moody's Method Evaluation Approach

Experimental approaches are used to evaluate the success of using a method in practice. Our extension of SecEval to express Moody's concepts (cf. section 2.2.2) is shown in figure 3.11: We introduce a `MoodyExperiment` class as a descendant of `UsedResource` – a class from SecEval's Data Analysis model. It is connected to at least one method (from the Security Context model) and vice versa, which is depicted by the upper blue association. The experiment uses the method on at least one example. Observed and surveyed participants can either be practitioners or students. `ParticipantSurveys` are the `Source` of information for the research question the experiment is conducted for. Two types of experiments exist: laboratory experiments with student participants and field experiments with practitioners. Multi-inheritance is used to be able to describe an experiment not only as a resource for obtaining data, but also as a method, e.g., by relating it to other methods. For example, a concrete experiment instance might relate to a method instance like "acceptance testing".

The lower blue association allows to access the resulting information, grouped by Moody et al. [136] into performance, perception, intensions and behavior, as depicted in the lower half of figure 3.11:

- "Actual Efficiency: the effort required to apply a method.
- Actual Effectiveness: the degree to which a method achieves its objectives.
- Perceived Ease of Use: the degree to which a person believes that using a particular method would be free of effort.
- Perceived Usefulness: the degree to which a person believes that a particular method will be effective in achieving its intended objectives.
- Intention to Use: the extent to which a person intends to use a particular method.
- Actual Usage is the extent to which a method is used in practice" [136].

We could have used instances of `Criterion` instead of adding classes for each resulting group. However, for many experiments carried out following Moody's approach, an advantage of classes is that their predefined structure facilitates information retrieval, contrary to instances that would need to reestablish this structure for each experiment.

Purple-colored associations show where the resulting information originates from: Performance is based on observing laboratory experiments that were not conducted with practitioners,

[10]CVSS. https://nvd.nist.gov/cvss.cfm
[11]Microsoft Threat Rating. https://msdn.microsoft.com/en-us/library/aa302419.aspx#c03618429_011

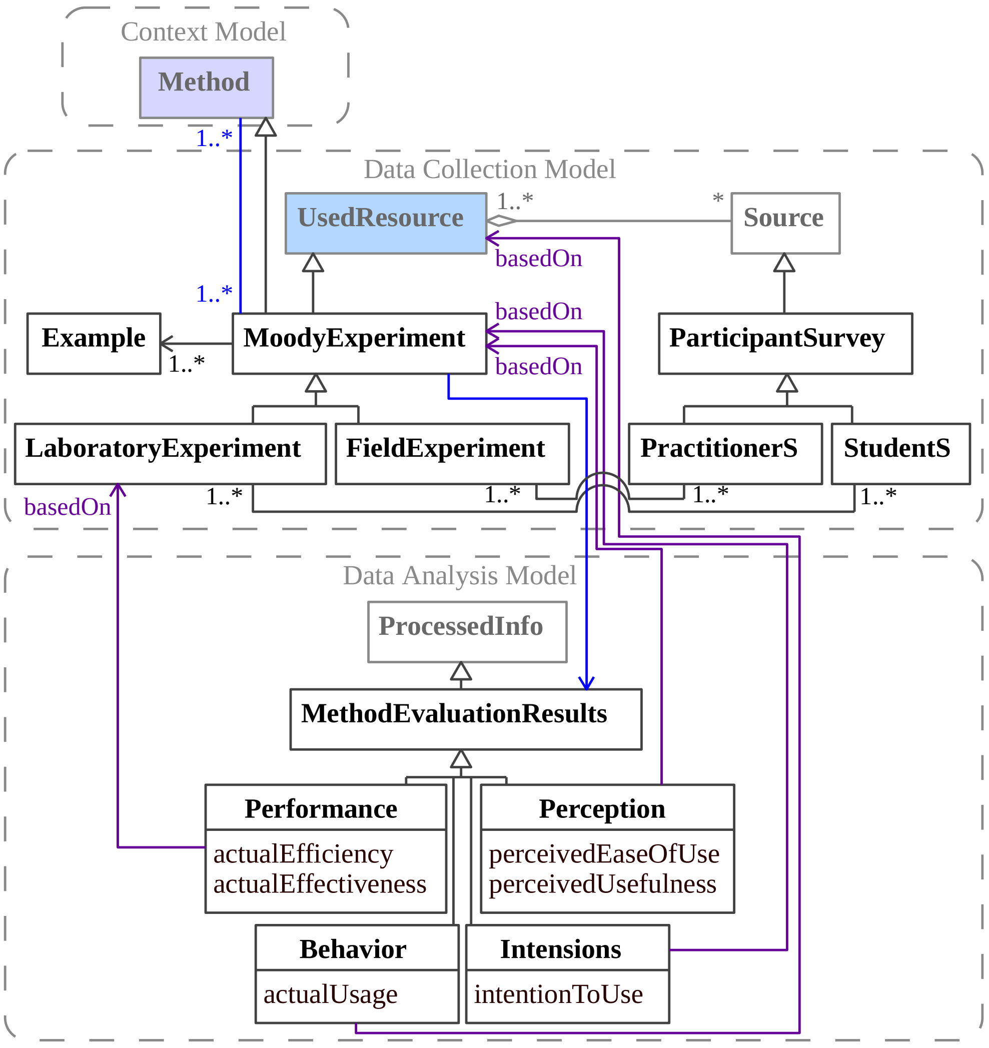


Figure 3.11: Method extension using Moody's method evaluation approach

whereas perception and intention to use is based on (surveys subsequent to) laboratory as well as field experiments. Actual usage is typically determined from another `UsedResource` than from experiments, e.g., from large-scale surveys. In the end, the average value of the subclasses' attributes of `MethodEvaluationResults` can be used as final evaluation result for the method under test and can help to categorize methods using the `Category` class depicted in figure 3.4.

3.3 Guided Interview

To validate the relations and properties of security knowledge objects is challenging, because many different areas of expertise are needed. We had the opportunity to conduct a so-called "guided interview" with partners of the EU project NESSoS, who encompass the broad area of secure software development. A Guided Interview is "a one-on-one directed conversation with an individual that uses a pre-determined, consistent set of questions but allows for follow-up questions and variation in question wording and order."[12] In 2013, we hold this kind of interview in a slightly modified way: first we explained our basic model (especially the basic Security Context model). Second, we handed out a description of the draft version of SecEval and a questionnaire, which can be found in appendix B. Finally, 14 international senior researchers, who are experts in different areas of security engineering, gave us feedback.

The answers and discussions helped us to improve SecEval. Among other changes, further attributes were added, as e.g., `canBeUsedInteractively` and `canBeUsedAutonomously` to express if tools support an interactive or batch mode. Some classes and enumerations were split to emphasize the idea of separation of concerns. For example, the enumeration `MaturityLevel` was split into `TechnicalMaturity` and `Adoption`. In addition, we extended SecEval for risk rating and experimental approaches. A closer view on resulting changes can be found in [42, chapter 3.3].

3.4 Case Study: Web Vulnerability Scanning

Web applications are the focus of many attacks. Thus, many methods such as "penetration testing" or "vulnerability scanning" are used to identify security flaws. These methods are supported by many commercial and open-source tools and it is not easy to decide which one is the more suitable for the tests to be performed. In this section, we use our SecEval approach to evaluate vulnerability scanners for web applications. Two bachelor's theses [188, 122], supervised by the author, offered the initial data basis for our SecEval model examples in 2013. Note that this case study is an example of an evaluation, as depicted in the introduction (cf. figure 1.3, upper arrow).

3.4.1 Data Collection

According to the SecEval approach, the first step consists of specifying the data that should be collected. This is done by an instance model as shown in figure 3.12, which depicts instances of the classes depicted in figure 3.8. For example, instances of the class `ResearchQuestion` define two research questions, a high-level and a concrete one. We used identical background colors for

[12]Education dictionary. http://www.mondofacto.com/facts/dictionary?guided+interview

instances of the same classes and omitted all `name` attributes in case a name (e.g., `p3`) is given in the header of an instance.

Research question `q1` ("Which security-related tools and methods are available and how do they compare?", cf. figure 3.12) is very general. In the first process phase `p1`, 13 methods and 18 tools were selected [188]. More detailed information was gathered in the second process phase `p2` about: vulnerability scanning, penetration testing, fuzzing and the classification into black-, grey-, and white-box testing. Examples for tools are WSFuzzer, X-Create and WS-Taxi, just to mention a few.

For recording our information sources, we used a table which contains the name of a method / tool and URLs from which information should be extracted in the data analysis phase.

The research question we focus on in this section is depicted in figure 3.12 as instance called `q2` ("Which vulnerability scanners are available for testing security features of web applications?"). It is a typical question which could be asked by security engineers working in a company. The "sources" (i.e., tools) we selected for analysis were [122]:(a) Acunetix Web Vulnerability Scanner[13], (b) Netsparker[14], (c) Burp Scanner[15], (d) Wapiti[16], (e) Arachni[17], (f) Nessus[18], (g) Nexpose[19] and (h) Nikto[20].

The instance `experienceWithTestScenario` describes how the data is gathered by testing the vulnerability scanners. It is worth mentioning that SecEval does not impose the completion of the data collection phase before the data is analyzed. This means that the tests were partly executed on tools which were later classified as inappropriate. This becomes clear when we think of how evaluation works in practice: sometimes we have to collect a bunch of data before we observe information which, e.g., leads to the exclusion of a tool from the result set.

3.4.2 Data Analysis

In the analysis phase for question `q2` the analysis strategy is defined so that a filter enforces our requirements (cf. value of attribute `limitation` of `q2` in figure 3.12). Figure 3.13 depicts instances of SecEval's Data Analysis model we introduced in figure 3.9.

Before going into detail about particular results of our experiments, we first take a look at the overall result regarding our research question `q2`. Figure 3.13 thus depicts an instance of the class `ProcessedInfo`, which is called `weightedResultValues`.

Only four tools passed our filter: Arachni and Nikto, which provide command-line interfaces and Nessus and Nexpose, which also provide web interfaces. From our list of tools from above, the trial of (a) only allows to scan predefined sites. Tools (b) and (c) do not support a command line or web interface in the versions that are free. A run of tool (d) on our test target Multidae[21] with Metasploitable[22] took six hours.

[13]Acunetix. http://www.acunetix.com
[14]Netsparker. https://www.mavitunasecurity.com/netsparker
[15]Burp Scanner. http://portswigger.net/burp/scanner.html
[16]Wapiti. http://www.ict-romulus.eu/web/wapiti
[17]Arachni. http://www.arachni-scanner.com
[18]Nessus. http://www.tenable.com/de/products/nessus
[19]Nexpose. https://www.rapid7.com/products/nexpose
[20]Nikto. http://www.cirt.net/Nikto2
[21]NOWASP (Mutillidae). http://sourceforge.net/projects/mutillidae
[22]Metasploitable. http://www.offensive-security.com/metasploit-unleashed/Metasploitable

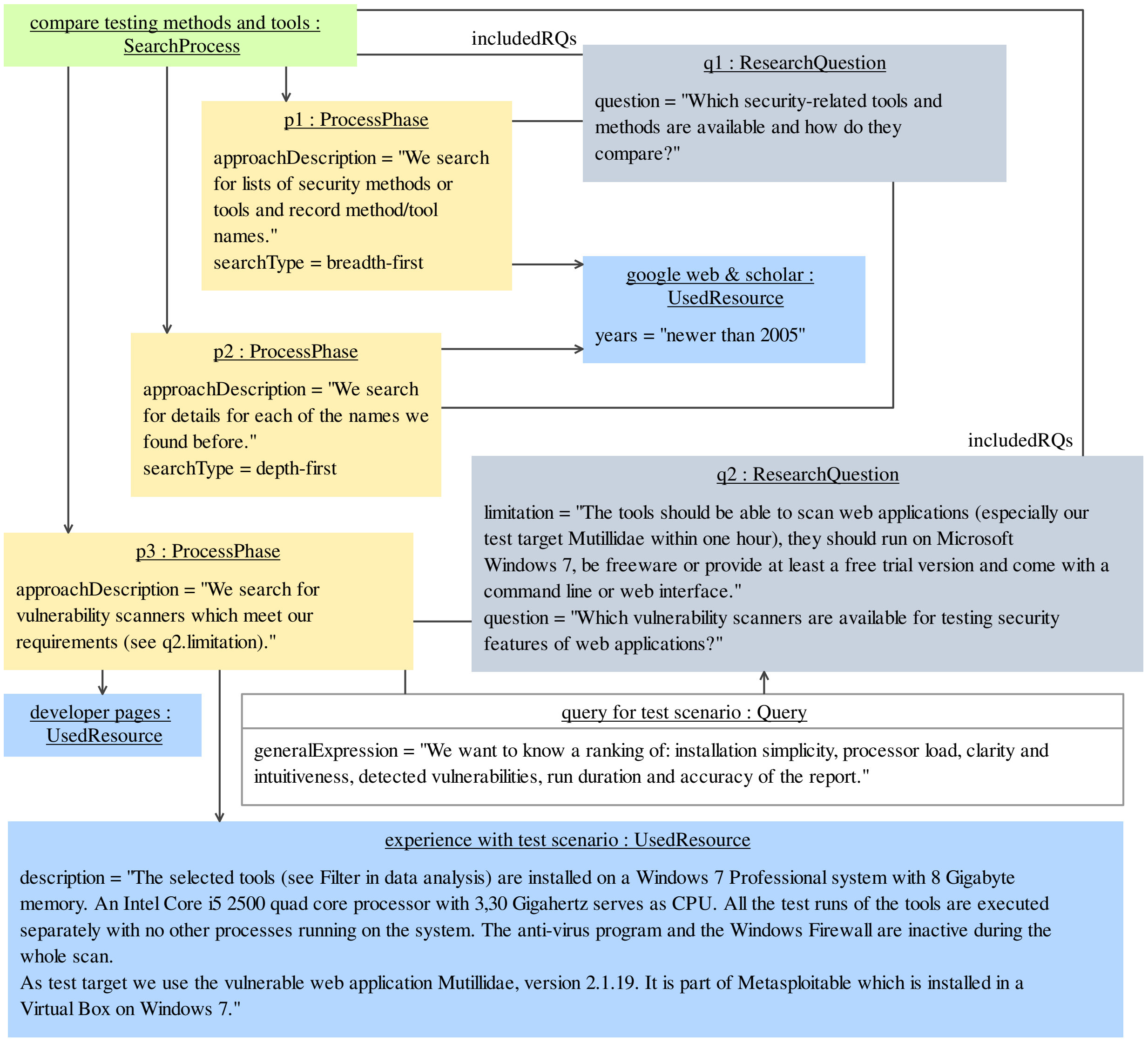

compare testing methods and tools : SearchProcess
includedRQs
q1 : ResearchQuestion
question = "Which security-related tools and methods are available and how do they compare?"
p1 : ProcessPhase
approachDescription = "We search for lists of security methods or tools and record method/tool names."
searchType = breadth-first
google web & scholar : UsedResource
years = "newer than 2005"
p2 : ProcessPhase
approachDescription = "We search for details for each of the names we found before."
searchType = depth-first
includedRQs
q2 : ResearchQuestion
limitation = "The tools should be able to scan web applications (especially our test target Mutillidae within one hour), they should run on Microsoft Windows 7, be freeware or provide at least a free trial version and come with a command line or web interface."
question = "Which vulnerability scanners are available for testing security features of web applications?"
p3 : ProcessPhase
approachDescription = "We search for vulnerability scanners which meet our requirements (see q2.limitation)."
developer pages : UsedResource
query for test scenario : Query
generalExpression = "We want to know a ranking of: installation simplicity, processor load, clarity and intuitiveness, detected vulnerabilities, run duration and accuracy of the report."
experience with test scenario : UsedResource
description = "The selected tools (see Filter in data analysis) are installed on a Windows 7 Professional system with 8 Gigabyte memory. An Intel Core i5 2500 quad core processor with 3,30 Gigahertz serves as CPU. All the test runs of the tools are executed separately with no other processes running on the system. The anti-virus program and the Windows Firewall are inactive during the whole scan.
As test target we use the vulnerable web application Mutillidae, version 2.1.19. It is part of Metasploitable which is installed in a Virtual Box on Windows 7."

Figure 3.12: Vulnerability scanning case study: data collection

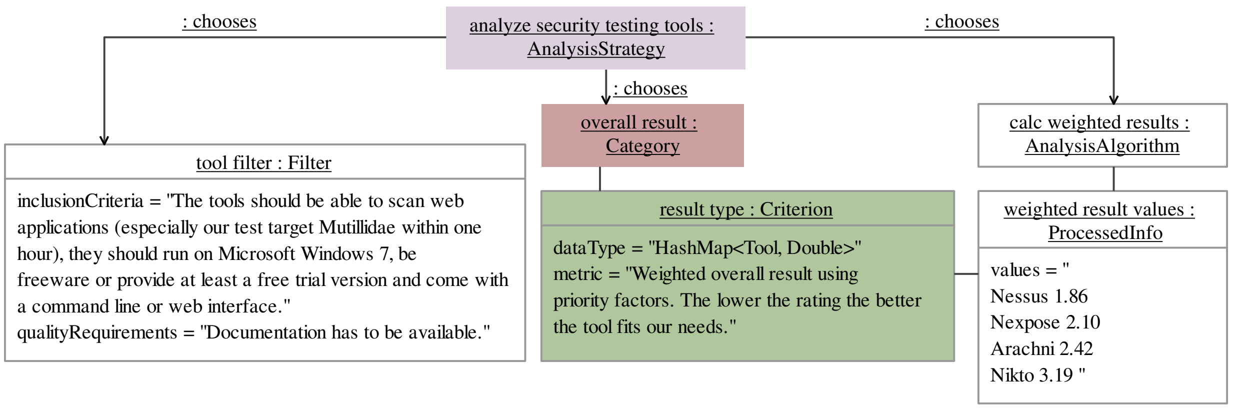

: chooses
analyze security testing tools : AnalysisStrategy
: chooses
: chooses
overall result : Category
calc weighted results : AnalysisAlgorithm
tool filter : Filter
inclusionCriteria = "The tools should be able to scan web applications (especially our test target Mutillidae within one hour), they should run on Microsoft Windows 7, be freeware or provide at least a free trial version and come with a command line or web interface."
qualityRequirements = "Documentation has to be available."
result type : Criterion
dataType = "HashMap<Tool, Double>"
metric = "Weighted overall result using priority factors. The lower the rating the better the tool fits our needs."
weighted result values : ProcessedInfo
values = "
Nessus 1.86
Nexpose 2.10
Arachni 2.42
Nikto 3.19 "

Figure 3.13: Vulnerability scanning case study: data analysis – results

Apart from information available online, we experimented with the tools that passed the filter, in order to obtain data for our tool evaluation (`q2`). We evaluated the following criteria (and weighted them as indicated in the brackets, cf. `queryForTestScenario`):

- Installation simplicity (0.5)
  Do any problems occur during installation?
- Costs (1)
  How much do the tool cost? Is it a one-time payment or an annual license?
- Processor load (1)
  How high is the CPU load while running the scanner
- Clarity and intuitiveness (1)
  Is the tool easy to understand, clearly structured and user-friendly
- Run duration (1)
  How long does a scan take?
- Quality of the report (2)
  How detailed is the report of the scan? Which information does it contain?
- Number of detected vulnerabilities (4)
  How many vulnerabilities does the tool detect on our test environment?

We selected these criteria, because they cover the process of working with a vulnerability scanner: first the costs when buying it and the installation effort needed in the beginning, second the intuitiveness, e.g. for setting up a test, third the processor load and run duration when executing a test and finally the quality of the report and the number of detected vulnerabilities.

As we can see in figure 3.13, an algorithm is involved, which calculates results according to our weighting. The consequent ranking is depicted in figure 3.14.

Lower factors of a criterion's priority denote that we consider the criterion less important. Table 3.1 contains the measured results as well as the average[23] and weighted[24] results.

[23]AVG: average
[24]WAVG: weighted average according to ratings

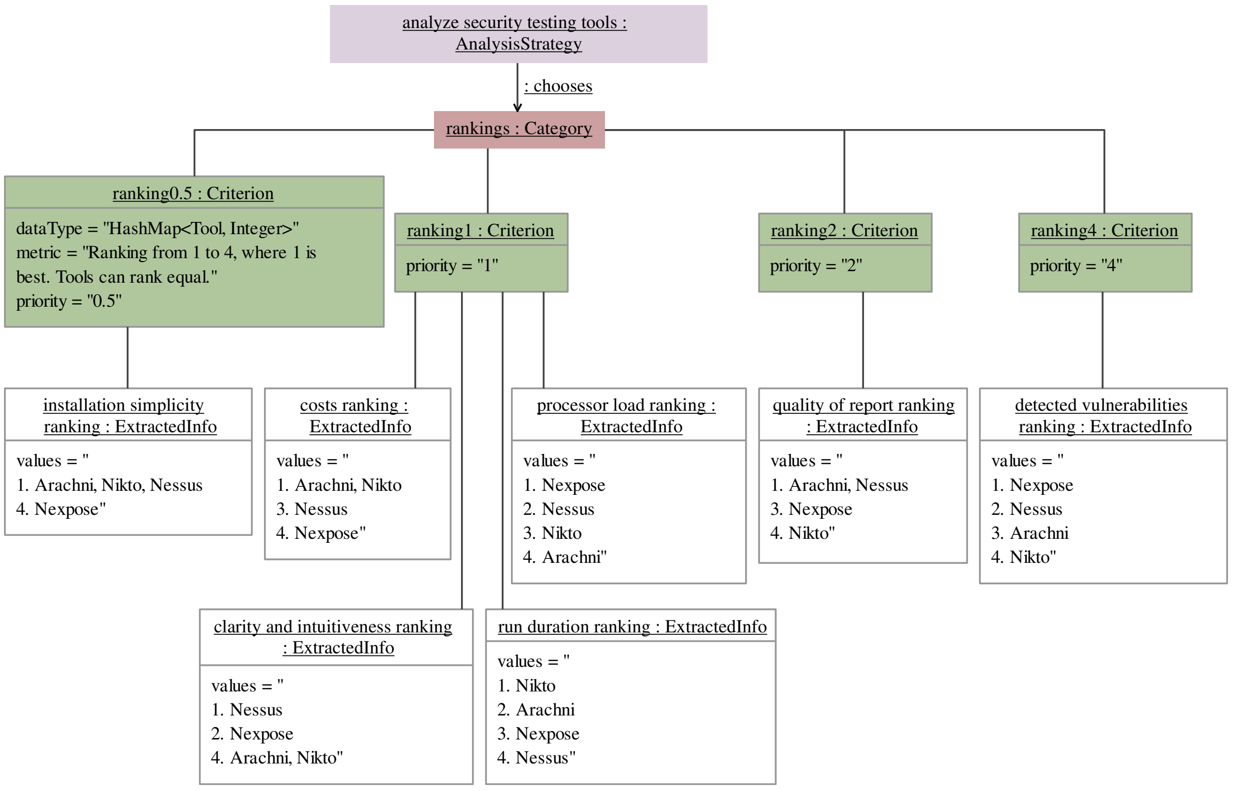

analyze security testing tools : AnalysisStrategy
: chooses
rankings : Category
ranking0.5 : Criterion
dataType = "HashMap<Tool, Integer>"
metric = "Ranking from 1 to 4, where 1 is best. Tools can rank equal."
priority = "0.5"
ranking1 : Criterion
priority = "1"
ranking2 : Criterion
priority = "2"
ranking4 : Criterion
priority = "4"
installation simplicity ranking : ExtractedInfo
values = " 1. Arachni, Nikto, Nessus 4. Nexpose"
costs ranking : ExtractedInfo
values = " 1. Arachni, Nikto 3. Nessus 4. Nexpose"
processor load ranking : ExtractedInfo
values = " 1. Nexpose 2. Nessus 3. Nikto 4. Arachni"
quality of report ranking : ExtractedInfo
values = " 1. Arachni, Nessus 3. Nexpose 4. Nikto"
detected vulnerabilities ranking : ExtractedInfo
values = " 1. Nexpose 2. Nessus 3. Arachni 4. Nikto"
clarity and intuitiveness ranking : ExtractedInfo
values = " 1. Nessus 2. Nexpose 4. Arachni, Nikto"
run duration ranking : ExtractedInfo
values = " 1. Nikto 2. Arachni 3. Nexpose 4. Nessus"

Figure 3.14: Vulnerability scanning case study: data analysis – ratings

Tool	Inst.	Costs	CPU	Clarity	Time	Vuln.	Rep.	AVG[23]	WAVG[24]
Nessus	1	2	2	1	4	1	2	1,86	1,86
Arachni	1	1	4	4	2	1	3	2,29	2,42
Nexpose	4	4	1	2	3	3	1	2,57	2,10
Nikto	1	1	3	4	1	4	4	2,57	3,19

Table 3.1: Case Study: Final Tool Ranking (adapted from [122])

3.4.3 Security Context

We modeled the context of vulnerability scanning of web applications and two of the tested tools, namely Nessus and Nikto in figure 3.15, which is an instance diagram of the Security Context model.

Note that from the criteria we used for evaluating the tools, only "costs" is non-experimental. Thus, costs can directly be added to `Tool` instances in figure 3.15. Except costs, all criteria are rather subjective, which means that they depend on our experimental setup. Consequently, they have to remain connected to the Data Collection and Analysis models. If someone wants to know more about a tool depicted in the Security Context model, searching research questions that use this tool (cf. dependency ≪use≫) lead to associated evaluations.

The depicted vulnerabilities are the top 3 from OWASP's top 10 project 2013 [161]. Further vulnerabilities are modeled in the course of presenting our Secure Web Applications' Ontology in the next chapter.

3.5 Summary and Related Work

SecEval, our conceptual framework for the development and evaluation of research questions related to secure software engineering provides an answer to our first research question (RQ1)[25]. This chapter presents SecEval. It consists of two main parts: one to describe concepts, their attributes and relations and one for evaluation, i.e., for answering knowledge questions that are related to these concepts, like comparisons of security engineering methods or tools. We validated the soundness of our approach by a guided interview and by an evaluation case study.

Structure of security concepts. The Security Context model comprises the following key concepts: assets, methods, tools, security properties, vulnerabilities and threats (so-called knowledge objects). A distinguishing characteristic of our evaluation framework SecEval is the refinement of methods and tools based on the phases of the SDLC. Additionally, for describing a knowledge object in detail, the model provides typed attributes.

Although the CBK [49] can be seen as a starting point for SecEval's Security Context model, we solve some issues in another way. For example, we do not consider standards as first-class citizens, because almost everything can be standardized. In addition, we aggregate the concepts of technique and method, as an instance model of SecEval can immediately show whether actions (in our case called steps) are defined. In contrast to the CBK, SecEval focuses explicitly on security-related features by providing a fine-grained model. An overview of other books of knowledge can be found in [76]. In contrast to our work they are not based on a detailed structure of concepts with attributes, and their relations. However, elements in metamodels of security ontologies usually feature relations, but few or no attributes, as e.g., [83, 101, 201]. Over the last years, it turned out that SecEval's novel structure for connecting and describing security-related knowledge objects serves as a reliable basis for method and tool evaluation as well as for the ontology SecWAO.

[25]Which key concepts emerge while developing secure software? Which properties do these concepts have and how are they related to each other? How can this knowledge be applied to evaluate concepts? (cf. section 1.2)

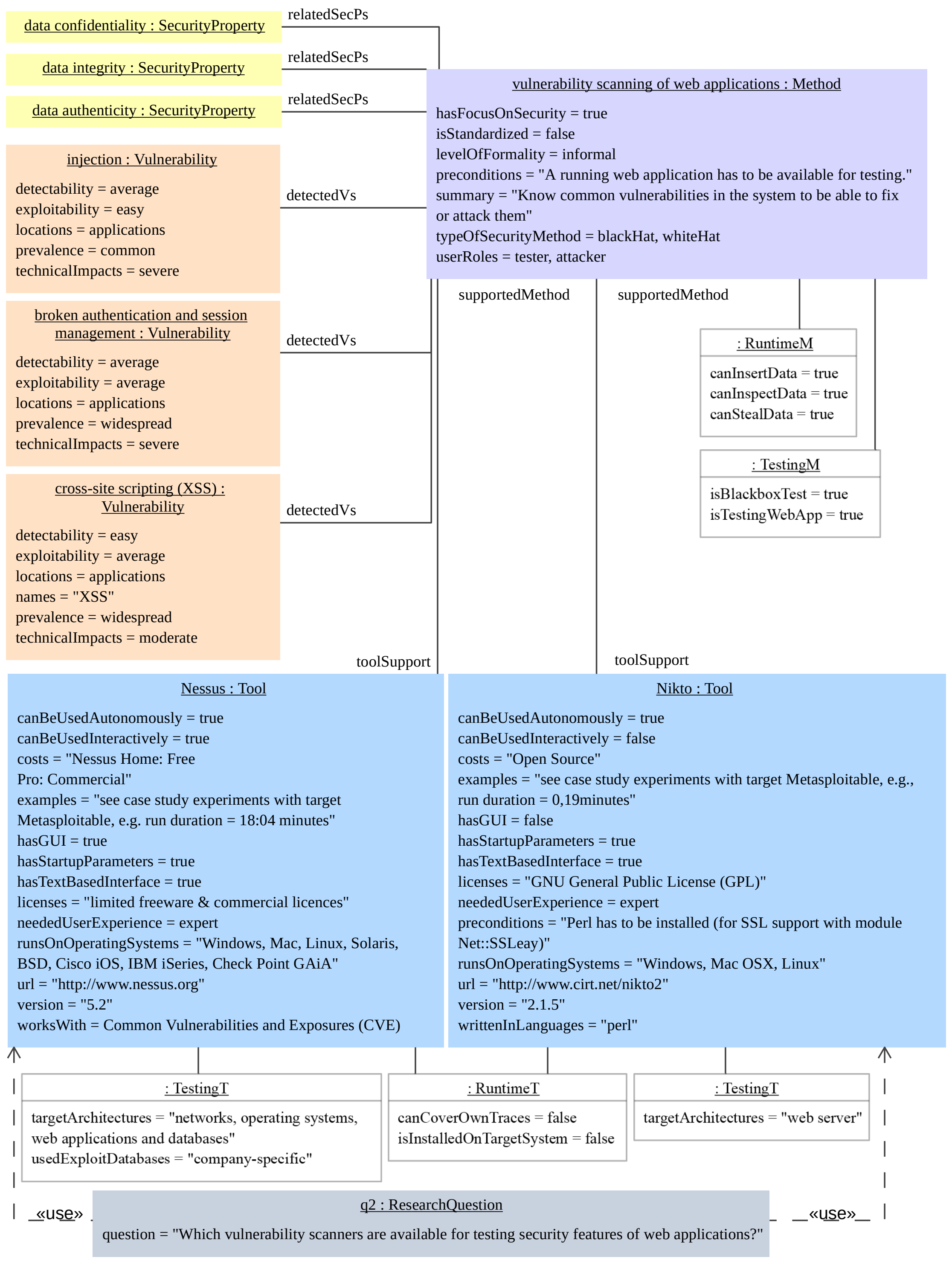

data confidentiality : SecurityProperty
relatedSecPs
data integrity : SecurityProperty
relatedSecPs
data authenticity : SecurityProperty
relatedSecPs
vulnerability scanning of web applications : Method
hasFocusOnSecurity = true
isStandardized = false
levelOfFormality = informal
preconditions = "A running web application has to be available for testing."
summary = "Know common vulnerabilities in the system to be able to fix or attack them"
typeOfSecurityMethod = blackHat, whiteHat
userRoles = tester, attacker
injection : Vulnerability
detectability = average
exploitability = easy
locations = applications
prevalence = common
technicalImpacts = severe
detectedVs
broken authentication and session management : Vulnerability
detectability = average
exploitability = average
locations = applications
prevalence = widespread
technicalImpacts = severe
detectedVs
cross-site scripting (XSS) : Vulnerability
detectability = easy
exploitability = average
locations = applications
names = "XSS"
prevalence = widespread
technicalImpacts = moderate
detectedVs
supportedMethod
supportedMethod
: RuntimeM
canInsertData = true
canInspectData = true
canStealData = true
: TestingM
isBlackboxTest = true
isTestingWebApp = true
toolSupport
toolSupport
Nessus : Tool
canBeUsedAutonomously = true
canBeUsedInteractively = true
costs = "Nessus Home: Free Pro: Commercial"
examples = "see case study experiments with target Metasploitable, e.g. run duration = 18:04 minutes"
hasGUI = true
hasStartupParameters = true
hasTextBasedInterface = true
licenses = "limited freeware & commercial licences"
neededUserExperience = expert
runsOnOperatingSystems = "Windows, Mac, Linux, Solaris, BSD, Cisco iOS, IBM iSeries, Check Point GAiA"
url = "http://www.nessus.org"
version = "5.2"
worksWith = Common Vulnerabilities and Exposures (CVE)
Nikto : Tool
canBeUsedAutonomously = true
canBeUsedInteractively = false
costs = "Open Source"
examples = "see case study experiments with target Metasploitable, e.g., run duration = 0,19minutes"
hasGUI = false
hasStartupParameters = true
hasTextBasedInterface = true
licenses = "GNU General Public License (GPL)"
neededUserExperience = expert
preconditions = "Perl has to be installed (for SSL support with module Net::SSLeay)"
runsOnOperatingSystems = "Windows, Mac OSX, Linux"
url = "http://www.cirt.net/nikto2"
version = "2.1.5"
writtenInLanguages = "perl"
: TestingT
targetArchitectures = "networks, operating systems, web applications and databases"
usedExploitDatabases = "company-specific"
: RuntimeT
canCoverOwnTraces = false
isInstalledOnTargetSystem = false
: TestingT
targetArchitectures = "web server"
«use»
q2 : ResearchQuestion
question = "Which vulnerability scanners are available for testing security features of web applications?"
«use»

Figure 3.15: Vulnerability scanning case study: instances of the Security Context model (excerpt)

Data collection and analysis. In addition to a customizable process model, SecEval defines class models for (a) data collection, i.e., planning and documenting research questions, search process phases, used resources, (concrete) queries on used resources and finally recording sources that are found; (b) analysis strategies that define analysis algorithms, categories and filters for analyzing these sources. Resulting information might then be stored in an instance model of the Security Context model.

The Data Collection and Analysis models are based on Kitchenham's approach of structured literature reviews [113]. Both models are not security-specific (contrary to the Security Context model), which means they can also be applied to other domains. In contrast to Kitchenham's approach, our data collection process is iterative, and more specific for a chosen domain as we specify a detailed structure of the context for which we pose the research questions.

The conceptual framework INCAMI [156] targets the quality of web applications and not the evaluation of mechanisms for engineering secure software, as SecEval does. Consequently, they focus on requirements of a web application and on measures that quantify the quality of their implementation. Nevertheless, INCAMI inspired us to use UML class diagrams for systematically documenting the evaluation process and to avoid adding too many details to the basic SecEval models.

Validation. During the development of SecEval, we conducted a guided interview to get feedback that helped us to improve the models and to get an impression of how our approach was received. In addition, we executed a case study about methods and tools from the area of security focusing on a research question about the selection of vulnerability scanners for web applications. In the evaluation of web vulnerability scanners Nessus ranked first amongst Arachni, Nexpose and Nikto according to our requirements.

Outlook. For many purposes, SecEval's Security Context model can be used without any changes, be it for evaluation or as a basis for an ontology, as shown in the next chapter. That is why we are confident that we have chosen the most important elements to be included in the model by default. However, it is explicitly designed to be adapted to concrete needs to be as useful as possible for an intended purpose, as shown exemplarily for risk rating and method evaluation approaches.

For example, for the evaluation of commercial tools, it might be preferable to detail the `costs` for using a `Mechanism` (cf. figure 3.4), as the most convenient level of detail has to be defined individually for a set of research questions. Sometimes, the overall costs should be recorded; sometimes it is necessary to split the costs into cost to train personal, yearly license costs for using proprietary tools, etc. For the sake of clarity and to provide an orderly overview, we limited the number of attributes per element in the default model to less than a dozen.

Another possibility is to merge existing, specialized models with SecEval. For example, Reggio et al. [175] present a conceptual model for comparing modeling methods, defining concepts for "Notation", "Modeling Method" and "Tool Support" (cf. [175, fig. 1]). All three could easily be unified with the classes `Notation`, `Method` and `Tool` of the Security Context model, which would allow the seamless annexation of concepts they additionally specified, as e.g., "Intended Use of Models".

Chapter 4

SecWAO: A Secure Web Applications' Ontology

It is commonly known that most applications suffer from security holes that are sooner or later exploited. One reason is that for developers the term “security” is difficult to grasp. Many security properties exist and there are many methods to enforce them or to avoid implementing common vulnerabilities in applications. In a keynote talk in 2014, Elçi [79] presented existing security ontologies (cf. section 2.2.2) and noticed that many researchers focus either on web engineering or on security, but only few on both.

In this chapter, which is a revised and extended version of one of our papers [48], we use SecEval to build a domain ontology[1] for web application security, i.e., we instantiate SecEval's Security Context model to relate relevant assets, methods, tools, security properties, vulnerabilities and threats. This ontology provides a structured overview and it will serve as a basis for our modeling approach for secure web applications in part III, as already depicted in the introduction in figure 1.2.

The process of data collection and data analysis while working on SecWAO cannot be reconstructed in detail, since implicit knowledge became more and more explicit over a long period while studying literature, discussing with members of the NESSoS EU project as well as meeting with several security-related regional groups in Munich. The overall research question RQ2, can be described as stated in the introduction (cf. section 1.2): “How are assets, security engineering methods, notations, tools, security properties, vulnerabilities and threats related in the domain of web application security?”. At this point, we extend RQ2 by asking “Is SecEval's Security Context model powerful enough to express these relations?”.

We introduce SecWAO by example, before we detail the relationships between web-related security properties, methods, and vulnerabilities. Note that diagrams depict excerpts of the ontology by visualizing chosen perspectives. Finally, we present an implementation of a knowledge base that is based on SecEval. It can be used for describing and relating knowledge objects for evaluation purposes or for representing an ontology based on SecEval, like our ontology SecWAO.

[1]“Domain ontologies capture the knowledge valid for a particular type of domain” [82, p. 5]

4.1 Overview of SecWAO by Example

In the following, we illustrate relations of Cross-Site Scripting (XSS) to give an insight into SecWAO. Hence, we extend the small example that was initially presented in chapter 1. As already mentioned in the introduction, XSS is a kind of injection that aims at adding malicious script code – usually JavaScript – to a website so that the browser executes the code [195, p.24]. According to OWASP's Top 10 [161], it is the third riskiest web applications' vulnerability and the most widespread and according to Weinberger et al. [226] XSS protection is not sufficiently prevented by existing web frameworks.

In terms of SecEval, XSS is an instance of the class `Vulnerability` from the Security Context model (cf. section 3.2.1). The upper part of figure 4.1 depicts the main elements of this model and serves as a legend. Instances of these classes and associations are shown in the lower part, around the aforementioned instance named `Cross-Site Scripting (XSS)`. In UML, an instance is denoted by an underlined object name, followed by a colon and the name of the class that it instantiates. In this chapter, the second part is often hidden in the case that a legend provides distinctive features, like different shades of color.

In figure 4.1 SecWAO helps not only to express that XSS is a kind of injection but also that it is threatened by JavaScript code provided in a way to be executed[2]. In practice, the name of threats and vulnerabilities are often used for both, e.g. the instance of `Threat` could also be named "Cross-Site Scripting (XSS)". There are subtypes of XSS: reflected XSS executes code in a browser that has (most of the time involuntarily) been sent by a user. Stored XSS is delivered to all browsers that visit a page that contains vulnerable content, which has been stored at the server since a successful attack [171, p.10]. A vulnerability would be harmless if it does not jeopardize security properties. Weakening control flow integrity[3] can be especially harmful, because data security, as confidentiality of the user's data in the browser and the integrity of the data which is sent to the server are based on it. For example, JavaScript code injected into an online banking service might undermine the browser's Same-Origin-Policy[4] in order to secretly report the user's account balance to a third party. JavaScript code might also alter the amount of money after a user submitted a value for a bank transfer, which violates data integrity. In figure 4.1 we model general SecWAO assets as "user data" and "website in browser". However, assets can be finer grained in case developers want to extend SecWAO to model more concrete scenarios for their web application. General assets, which are not specific to web applications, can be found in [101, figure 3].

On the lower right of figure 4.1, methods to shield web applications from vulnerabilities like injection are depicted. In general, injection prevention uses data validation, preferably whitelisting, where allowed inputs are specified and different input is discarded. Other use cases require input sanitization by blacklisting, where developers struggle to filter all kinds of harmful inputs [167].

[2]Note that threats are methods, which means they can extend other threats. For example, a JavaScript injection might threaten the user's privacy and prepare identity theft in a concrete attack scenario.

[3]Control flow integrity is the property of software that restricts a user to execute functions in a predefined order, according to the program logic.

[4]The Same-Origin-Policy makes sure that "actors with differing origins are considered potentially hostile versus each other, and are isolated from each other" (http://www.w3.org/TR/html5/browsers.html#origin)

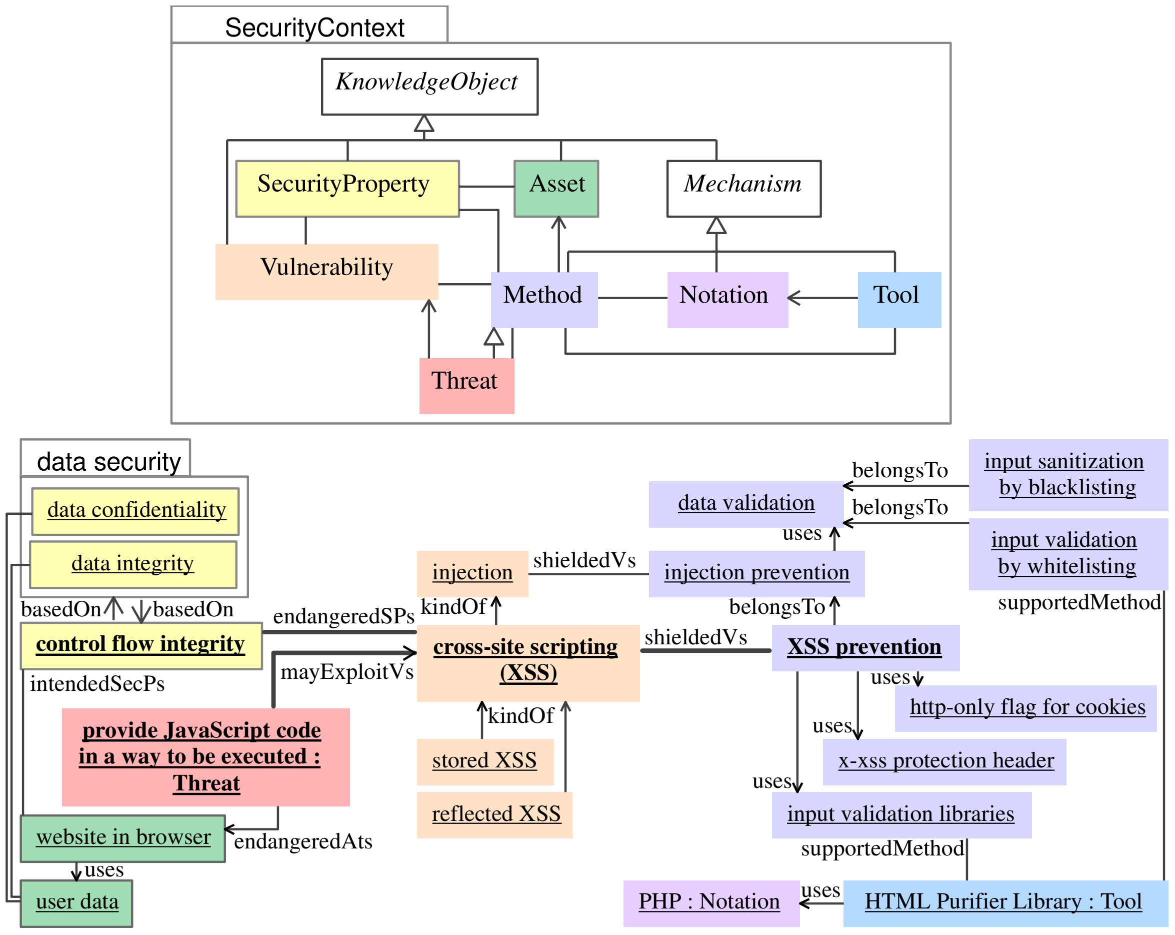


Figure 4.1: Upper part: SecEval's Security Context model, serving as a legend for the lower part: SecWAO example of relations regarding Cross-Site Scripting (XSS).

For web applications, data validation libraries for XSS-blacklisting exist. Additionally, the http-only flag for cookies[5] can be set and the x-xss protection header can advise browsers to turn on built-in XSS protection [215]. With instances of the Security Context model we can also specify that the HTML Purifier Library [229] is a data validation library written in PHP [173].

4.2 Security Properties

Security requirements consist of at least two elements: the asset that should be *secured* and the kind of security that "*secured*" refers to. The latter can be expressed in SecWAO as an instance of the class `SecurityProperty` from the Security Context model, which is the basis of SecWAO.

In this section, we interrelate common security properties by using instances of SecEval's association with the role `dependsOn` that belongs to the class `Security Property` (cf. figure 3.4). For us, "common" means that the security properties are mentioned in Wikipedia[6] and in several scientific publications, which was verified with Google Scholar[7]. In this section, we provide informal definitions for security properties, as the first goal of SecWAO is to provide an comprehensive view on knowledge objects and their relations to each other. This is why we also explain concepts in plain terms and do not refrain from citing Wikipedia in case it provides comprehensible definitions. In the following, we present SecWAO's main security properties, as depicted in figure 4.2, and some security properties in the context of key-agreement protocols, cryptosystems and digital signatures, as depicted in figure 4.3.

As shown in figure 4.2, availability[8] of a system depends on system integrity[9], because if e.g., an attacker has taken over a system, system integrity is not necessarily provided as the system might be shut down or destroyed to the attacker's liking. In case an attacker manages to violate control flow integrity, e.g., by using URLs that directly reference program functions, system integrity can be at stake. Vice versa, violating the control flow integrity of a program is possible if attackers control a system, as they can alter program code or the configuration of a web application firewall.

It is not surprising that data security (depicted as UML package in figure 4.2) depends on system integrity and control flow integrity and vice versa, especially as insecure data like disclosed configuration data might provide expedient information for attacks. Note that methods can shield from related vulnerabilities, as e.g., a JavaScript sandbox in a browser can prevent malicious code from endangering system integrity.

Data security refers to different security properties; the most common are data confidentiality, data integrity and data availability (bold in figure 4.2). When methods to ensure confidentiality and integrity are not implemented properly enough, all security properties that are based on them can be endangered. For example, disclosing confidential home addresses leads to a violation of

[5]"The HttpOnly attribute limits the scope of the cookie to http requests. In particular, the attribute instructs the user agent to omit the cookie when providing access to cookies via "non-http" APIs (such as a web browser API that exposes cookies to scripts)" [1].

[6]Wikipedia. https://www.wikipedia.org (English or German)

[7]Google Scholar. https://scholar.google.com

[8]Availability "refers to the ability to use the information or resource desired" [25, p.6].

[9]System integrity is a "condition of a system wherein its mandated operational and technical parameters are within the prescribed limits". (Wikipedia: System integrity)

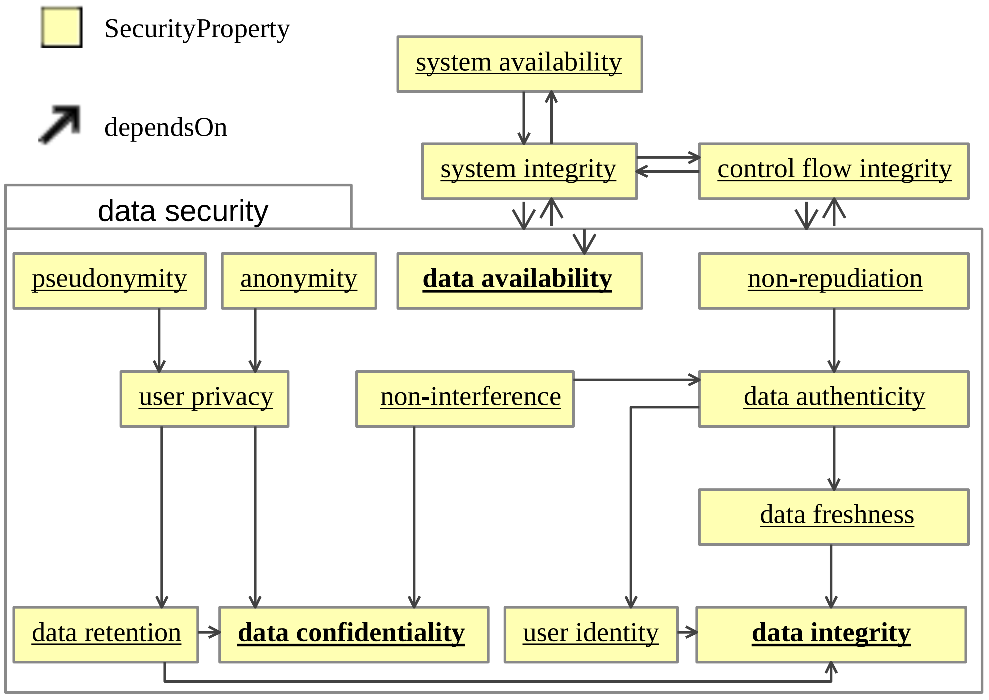


Figure 4.2: SecWAO security properties: overview

user privacy[10] as well as anonymity or pseudonymity, in case a web application allows users[11] to interact with a service without a (unique) name or by just using a nickname (cf. [172, p.614f]). Besides depending on confidentiality, privacy also depends on data retention[12]. Confidentiality, integrity and availability are often referred to as "CIA Triad", although many other security properties are required and realized these days. Note that in theory data that is not available by any means can be seen as secure. In practice, we see availability as a requirement that is crucial for making use of data, thus we model that data security in general "depends on" data availability.

Non-interference[13] is also based on confidentiality, as it must be possible to restrict read or write access to classified data. Additionally, it depends on data authenticity[14], which requires data freshness[15], data integrity and user identity[16].

Non-repudiation[17] needs data authenticity and is transitively based on the security properties on which data authenticity depends. The term "traceability" is sometimes used for weaker forms of non-repudiation like logging [13, p.12]. Non-repudiation can be seen as the security property required for accountability, which is "the requirement for actions of an entity to be traced uniquely to that entity"[18].

In the main ontology of Kim et al. [112], a list of security objectives is provided[19] that also has more elements than the CIA Triad. Our security properties share some of the concepts, although we differentiate between methods that help to reach security goals. For example, in SecWAO "replay prevention" is a method that helps to realize the security property of data freshness (cf. figure 4.5).

Security properties are closely related to assets they characterize, as e.g., the data, applications or systems that should be protected. Cherdantseva and Hilton examine how components of an information system (i.e., information, people, processes, hardware, software and networks) are related to common security properties in [54, table 1]. Although we decided for SecWAO to split integrity and availability into "system" and "data" (as these are the most common), another approach would be to always connect security properties with instances of SecEval's `Asset` class, to express, e.g., hardware or network integrity / availability.

In figure 4.3, we present general data security properties such as data confidentiality and data authenticity in the contexts of three different assets – corresponding to the boxes in the figure.

[10]Privacy "is the right to control who knows certain aspects about you, your communications, and your activities" [172, p.604].

[11]In this thesis, a "user" denotes a human user as well as actors that are controlled by a computer system.

[12]Data retention defines which information is stored and how long it will be kept.

[13]Non-interference "is a property that restricts the information flow through a system" [220, p.605]. This usually means that information and users are grouped by categories with different levels of security and information from one level can only affect information of other levels according to policies.

[14]Data authenticity defines that received data was send from users that are who they claim to be.

[15]Data freshness is given if data is up-to-date (and not a replay of data that was sent in the past).

[16](User) identity "is a set of information that distinguishes a specific entity from every other within a particular environment" [220, p.584].

[17]Non-repudiation "refers to an inability to disavow a previous agreement" [220, p.852]. Simplified: a user cannot deny to have sent or received a message at a given time with a given content.

[18]NIST SP 800-33. http://csrc.nist.gov/publications/nistpubs/800-33/sp800-33.pdf

[19]Security goals mentioned in [112] are confidentiality, availability, user authentication, message authentication, authorization, message integrity, key management, trust, host trust, replay prevention, covert channel prevention, separation, traffic hiding and anonymity.

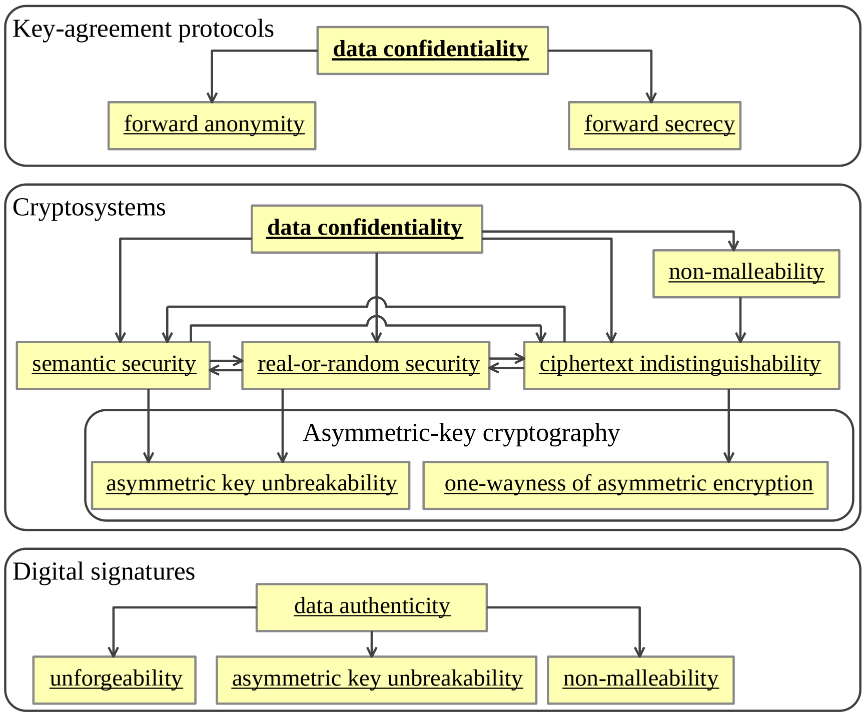


Figure 4.3: SecWAO security properties: details

In the context of protocols for agreeing on a key for future communication (key exchange), data confidentiality may depend on forward anonymity and forward secrecy. The former means that recorded data traffic and compromised (long-term) keys do not disclose the identities of the communication partners. The latter means that "disclosure of long-term secret keying material does not compromise the secrecy of the exchanged keys from earlier runs" [70, p.7]. In other words: session keys will not be revealed, even if all data traffic was recorded and long-term keys have been compromised after the sessions took place.

Confidentiality depends on properties of cryptosystems, as described in a Wikipedia article[20]. According to this article, semantic security[21], real-or-random security[22] and ciphertext indistinguishability[23] are equivalent and if they are broken, non-malleability[24] is also broken. Besides, typical security properties that are assumed to hold for asymmetric-key cryptography are key unbreakability[25] and one-wayness[26].

[20]German Wikipedia: Sicherheitseigenschaften kryptografischer Verfahren

[21]Semantic security means that attackers can derive nothing more than the length of an encrypted message.[20]

[22]Real-or-random security means that attackers cannot distinguish two encrypted messages, even if one of it encrypts a plaintext they provided.[20]

[23]Ciphertext indistinguishability means that attackers cannot distinguish pairs of cyphertexts, even if they know their plaintexts.[20]

[24]Non-malleability means that "given the ciphertext it is impossible to generate a different ciphertext so that the respective plaintexts are related" [72].

[25]Unbreakability means that attackers cannot calculate the private key from the public key.[20]

[26]One-wayness means that attackers cannot encrypt a given ciphertext.[20]

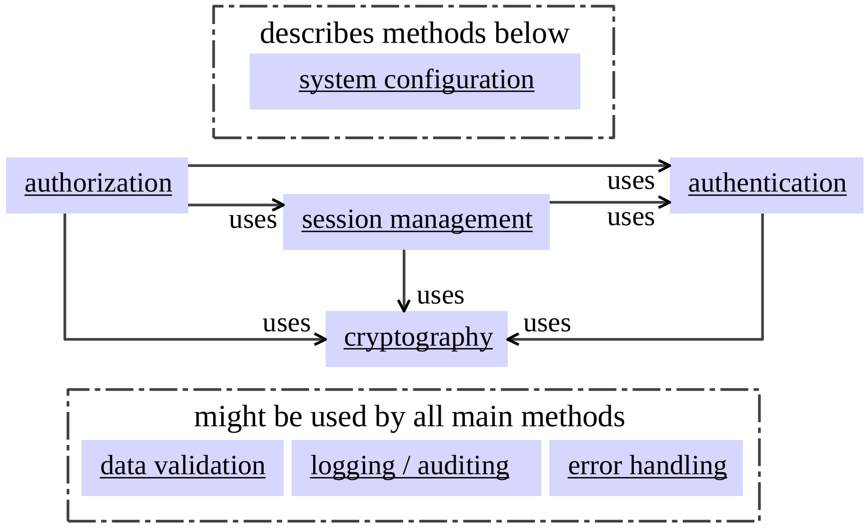


Figure 4.4: SecWAO: main methods

Security properties for digital signatures are depicted at the bottom of figure 4.3: Strong unforgeability[27] is based on existential unforgeability[28]. Asymmetric key unbreakability and non-malleability can also apply to digital signatures. In this context, non-malleability is referred to as not being able to create a second valid signature for a pair of a message and its valid signature.[20]

Other security features that can be modeled with SecWAO, but are not presented here are e.g., collision resistance, which is a property of cryptographic hash functions, "provable security", which means that security properties of an asset are mathematically proven or non-observability, which means that data is not only transmitted confidentially, but it also remains confidential whether information has been transmitted. Herzog et al. [101] specify goals that contain trust and correctness, which we omitted due to the vagueness of these terms.

4.3 Methods

Methods help to implement security properties for an asset. We group the main methods of SecWAO into three groups according to their mutual usage, as depicted in figure 4.4. In this section, we present SecWAO's main methods in general, before going into detail for cryptography, data validation, authentication, session management, authorization and other methods.

Authorization is a synonym for access control [220, p.2] and uses a successful authentication to identify users that request access, e.g., to an internal web page. This is represented in the

[27]Strong unforgeability "ensures the adversary cannot even produce a new signature for a previously signed message" [28].

[28]Existential unforgeability means that "an adversary who is given a signature for a few messages of his choice should not be able to produce a signature for a new message" [28].

diagram by instantiating the `Method`'s association with the role `uses` (cf. figure 3.7)[29]. The management of user sessions is also based on authentication, although anonymous sessions are possible that only require to identify, e.g., a cookie instead of authenticating a user. Cryptography aims at protecting “a secret from adversaries, interceptors, intruders, interlopers, eavesdroppers, or simply attackers, opponents, and enemies” [220, p.283]. As the aim of authorization is similar, it often uses cryptographic methods to enforce access control. For authentication cryptography is frequently employed to ensure confidentiality, integrity and freshness of requests in authentication protocols. Sessions require cryptographically strong identifiers that are unpredictable (cf. random number generators in figure 4.5).

Methods that are used by all depicted methods are placed at the bottom of figure 4.4, hiding all the linking `uses` arrows. For example, data validation has to implement error handling for coping with illegal inputs. Success messages (e.g., a successful authentication) as well as error messages or status messages (e.g., an expired user session) can be logged. We first thought about using “state handling” instead of “error handling”, because occurring errors might or might not lead to a change in the application's state. However, the relevance of handling errors deliberately outweighs the advantage of further abstraction in this security-focused ontology.

All tools that support the methods mentioned so far typically need to be configured. Otherwise, an authentication service might grant access to all users or an error message might disclose critical confidential information about a system. In short, system configuration is a method that relates to the secure employment of methods, i.e., it helps to describes how to adjust software and which hardware should be used. This is modeled using the `describes` association, although the links to all other methods are not shown in figure 4.4 for reasons of clarity.

In this thesis, we focus on technical methods, although we think that non-technical secure development methods are equally important. Examples are security-aware management (granting time and money for more secure implementations) and a security-aware software development life cycle, including e.g., security requirements elicitation, secure design methods, code reviews and penetration testing. Existing “applied security” literature typically includes a set of security principles[30] like simplicity, open design, minimum exposure, secure-by-default, fail securely etc. [13]. For example, secure-by-default and fail securely belongs to the methods “system configuration” and “error handling” in SecWAO.

4.3.1 Cryptography

The upper half of figure 4.5 details how cryptography relates to other methods. We use bold text in the object diagrams to identify methods that we think are central to recognize a certain set of methods, as e.g., methods related to cryptography. This set of methods support data integrity, authenticity, confidentiality and freshness, as can be seen at one glance in figure 4.5 (cf. instances of the class `SecurityProperty`). However, we just exemplarily depict links to security properties and assets, as we focus on methods and their interrelations.

[29]It is worth mentioning that role names, such as `uses`, are referring to a “common case” in our ontology. For `uses`, it means that something is “usually used”, but it does not state that it has to be used in any possible case.

[30]For a broad overview of practical security principles, the interested reader is referred to https://www.owasp.org/index.php/Category:Principle.

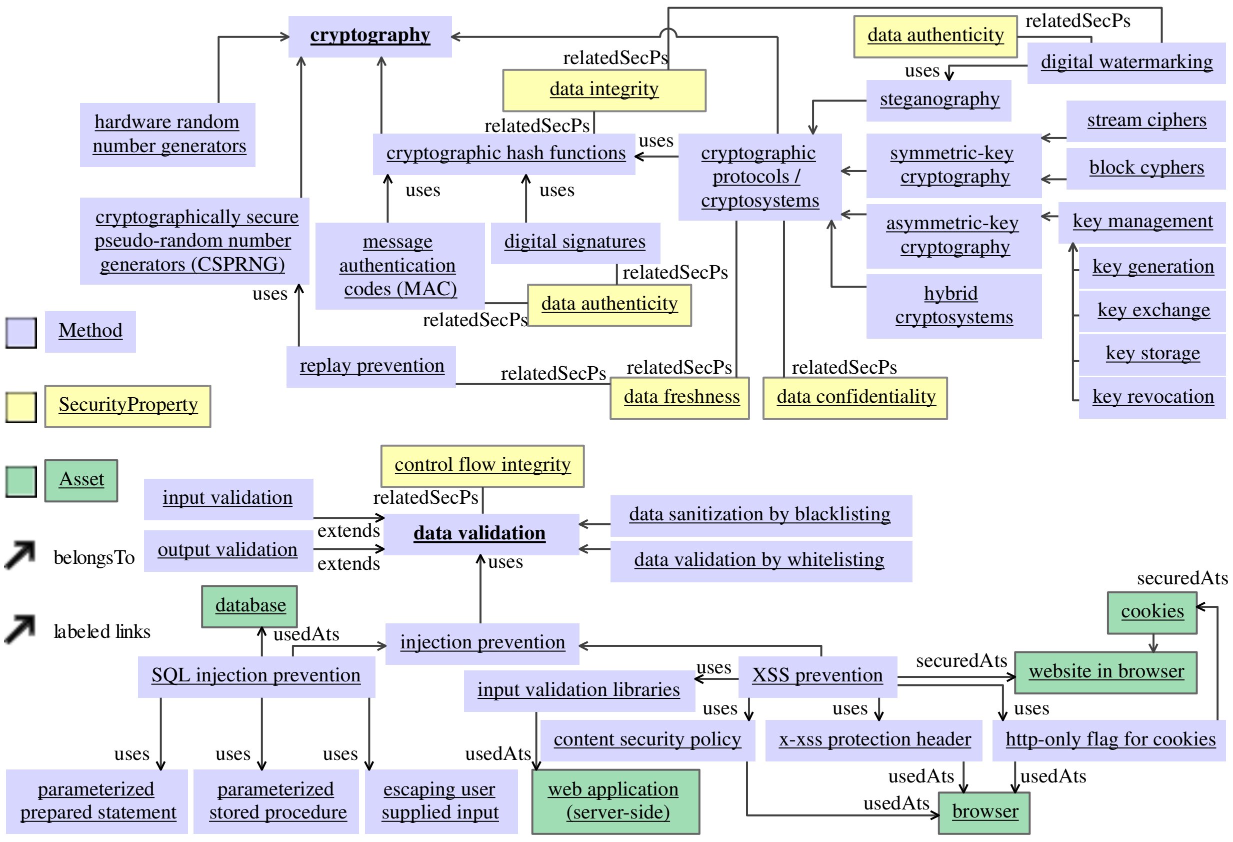


Figure 4.5: SecWAO methods: cryptography and data validation

A cryptosystem[31] can be symmetric, asymmetric or hybrid, which is determined by whether a single key is used for decryption and encryption, or a public and a private key are used, or an asymmetric key is used for encrypting a generated symmetric key that encrypts a message. Especially for asymmetric keys, key management is important, as such keys are often reused many times in contrast to symmetric ones.

Other methods that belong to cryptography are the mechanisms of random number generators and cryptographic hash functions (i.e., one-way hash functions). Random number generators can be implemented in hardware or software. Software implementations are so-called "secure pseudo-random number generators": algorithms that produce random numbers in a sequence that is determined by a seed. The seed has to be given as an input for the algorithm [220, p.995]. Generators are e.g., used for digital signatures that enforce data authenticity.

4.3.2 Data validation

Data validation – as known from the example of section 4.1 – is depicted on the bottom of figure 4.5. Input and output validation are kinds of data validation. The former denotes that data is validated when entering an application (possibly by a kind of firewall), whereas the latter

[31] A cryptosystem "is a system consisting of an encryption algorithm, a decryption algorithm, and a well-defined triple of text spaces: plaintexts, ciphertexts, and keytexts" [220, p.284].

refers to the validation just before data is handed over to a component where it could cause damage. Output validation has the advantage of knowing exactly how benign data should look like in a certain context and makes sure that data created by the application, as e.g., logfiles, are also taken into account.

Besides XSS prevention, e.g., supported by the Content Security Policy[32], other types of injection prevention exist. For example, database query injection (referred to as "SQL injection") can be avoided by using libraries for prepared statements. Prepared statements are available in most programming languages and they distinguish between user input[33] and SQL statements so that the former cannot influence the latter. Other options, as stored procedures[34] or escaping all user supplied input according to the database syntax that is used, are considered less secure, as a single careless mistake can bear a severe security flaw [169].

4.3.3 Authentication

Figure 4.6 focuses on authentication, including typical methods that are used for web applications, as registering for an account using common types of registration, ways to securely recover credentials and typical logout mechanisms. Authentication itself is often used as a synonym for checking the validity of the user's identification, which we can express by adding a link with the role `ambiguity` that belongs to `KnowledgeObject` (cf. figure 3.4). A less well-known method is to check if the user enters a password for accessing the application in a so-called "panic mode". This mode allows users from unsafe regions around the world to give away a valid password if threatened. This password permits an attacker to sign in to a web application that does not give rise to suspicion, while hiding personal user data and restricting access to critical functionality. That is why the panic mode is also related to state-based access control (cf. figure 4.7) [170].

4.3.4 Session Management

The upper part of figure 4.6 shows knowledge objects related to session management. On the upper left, common methods like starting or ending a session are depicted. In the case a critical error occurred during a session, it might be advisable to end the session. Note that sessions also exist for unauthenticated users; however, the session Identifier (ID) should be changed after authentication to avoid session fixation[35]. Other methods to prevent session takeover are depicted on the right: It can be helpful to enable users to list their active sessions so that they can invalidate

[32]Content Security Policy is "a mechanism web applications can use to mitigate a broad class of content injection vulnerabilities, such as Cross-Site Scripting. The server delivers the policy to the user agent via an http response header or an HTML meta element" [221].

[33]"User input" refers to any input sent by users or their devices, including input the user entered in a text field, cookies or protocol headers sent by a browser, etc.

[34]Stored procedures are located in the database and can be called from a client using parameters, which shifts the problem of securing SQL statements from the application to the stored procedure.

[35]Session fixation exploits cases where session IDs are not changed after the login, as an attacker can access a website to obtain an ID, trick a user to access and sign in to the same website using this ID (e.g. provided by a URL within an email) and can continue using the – now authenticated – session [168].

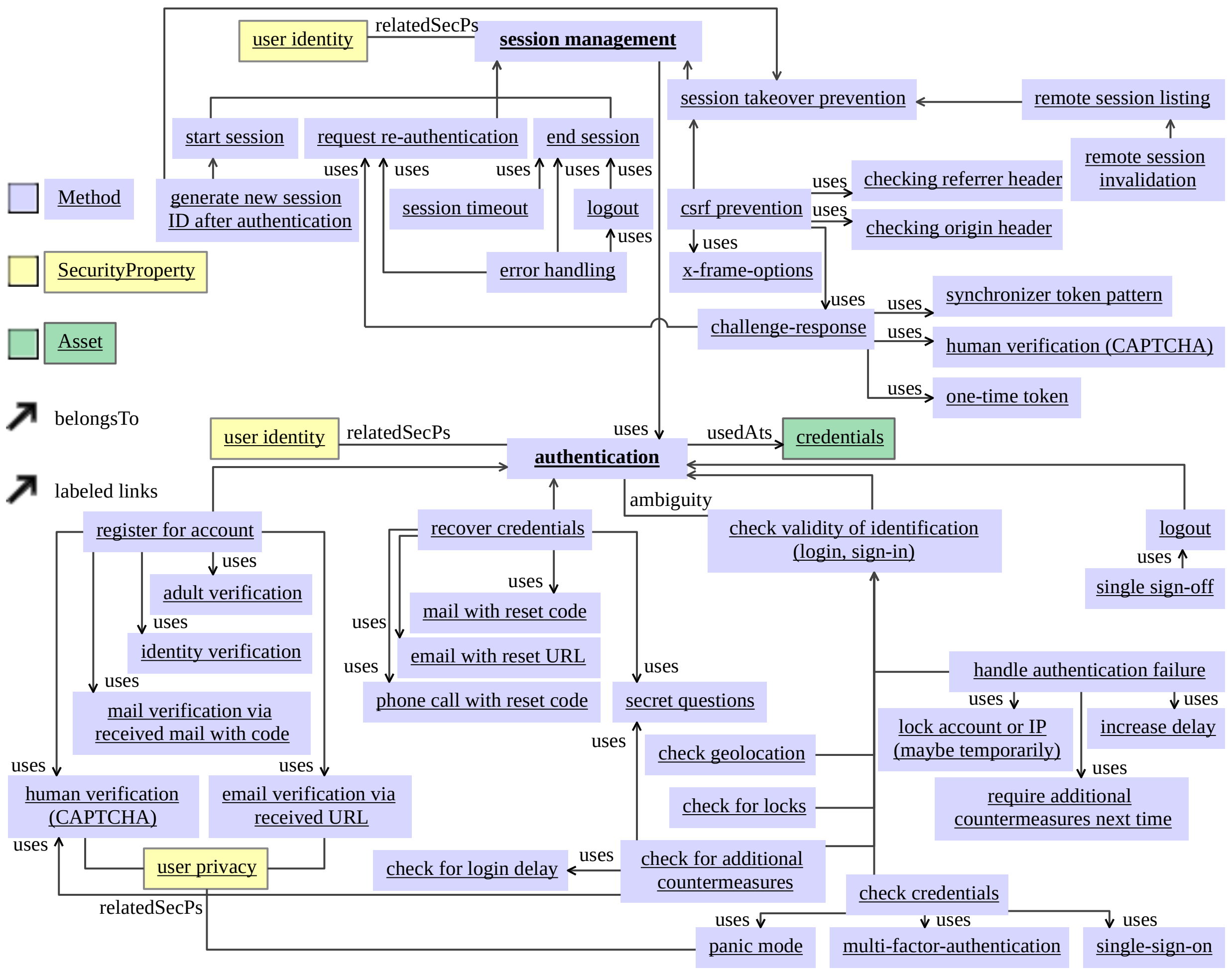


Figure 4.6: SecWAO methods: authentication and session management

them in case a device was stolen. Methods to prevent Cross-Site-Request-Forgery (CSRF)[36] are modeled according to their description in [164].

4.3.5 Authorization

Authorization defines an access source, an access target and actions that are permitted to be executed. In addition, figure 4.7 depicts an access control enforcement system that decides whether or not to permit a request for access, according to access control policies, which can be noted in languages as XACML [149]. These policies can be defined by a provider or by users, as expressed by the role `belongsTo` that is played by the category `access manager`. `Access control capabilities` describe common approaches used for authorization, as role-based access control (the user belongs to roles and access is specified for roles) or state-based access control (access control policies can, e.g., refer to the current time or the mode an application is in, like maintenance mode).

For web applications, common usages for cryptosystems are secure connections between a browser and a server to transmit confidential information. The protocol TLS [211] with its option for Perfect Forward Secrecy (PFS) (cf. security property "forward secrecy" in section 4.2) and the browser policy HTTP Strict Transport Security (HSTS)[37] are methods that are used in practice.

Besides, in figure 4.7 we use SecWAO to clarify the difference between `types of action`: functions of an application can be executed and data can be accessed. An example is DRM (Digital Rights Management), which aims at restricting copying, viewing or extracting information from files or videos. Data access can also be restricted by defining policies for CRUD (create, read, update, delete) on objects, as e.g., database records. Additionally, web applications focus on restricting navigational access. This allows to specify whether a user is allowed to navigate to a so-called "navigational node", i.e. a (part of) a web page. This avoids dead-ends, as it avoids navigating to a function a user is not allowed to access.

4.3.6 Logging, Error Handling and System Configuration

In figure 4.8 logging, error handling and system configuration are presented. They closely relate to system integrity and are used by many other methods (cf. figure 4.4). In the context of web applications, system integrity is also important for the client. Consequently, it is common courtesy to provide users with possibilities for download verification, especially when downloading programs. Digital signatures or cryptographic hashes can verify a download in case it was not transmitted by a protocol, as e.g., TLS that supports integrity.

Non-repudiation, beyond re-authentication before executing critical actions, has not commonly been realized for typical web applications so far. Digital signatures and logging (ideally by a third party) could ease traceability and forensics for critical actions like purchases, or changes in the configuration of safety-critical appliances that provide a web interface.

[36] A CSRF attack "causes a user's web browser to perform an unwanted action on a trusted site for which the user is currently authenticated" [164]. For example a user clicks on a link within a junk mail that uses an active session of an online shop to buy an unwanted product.

[37] HSTS enables "web sites to declare themselves accessible only via secure connections" [106], which prevents a man-in-the-middle from hijacking unencrypted requests (http) users sent to servers that usually redirect from unencrypted pages to an encrypted pages (https) *after* this first request.

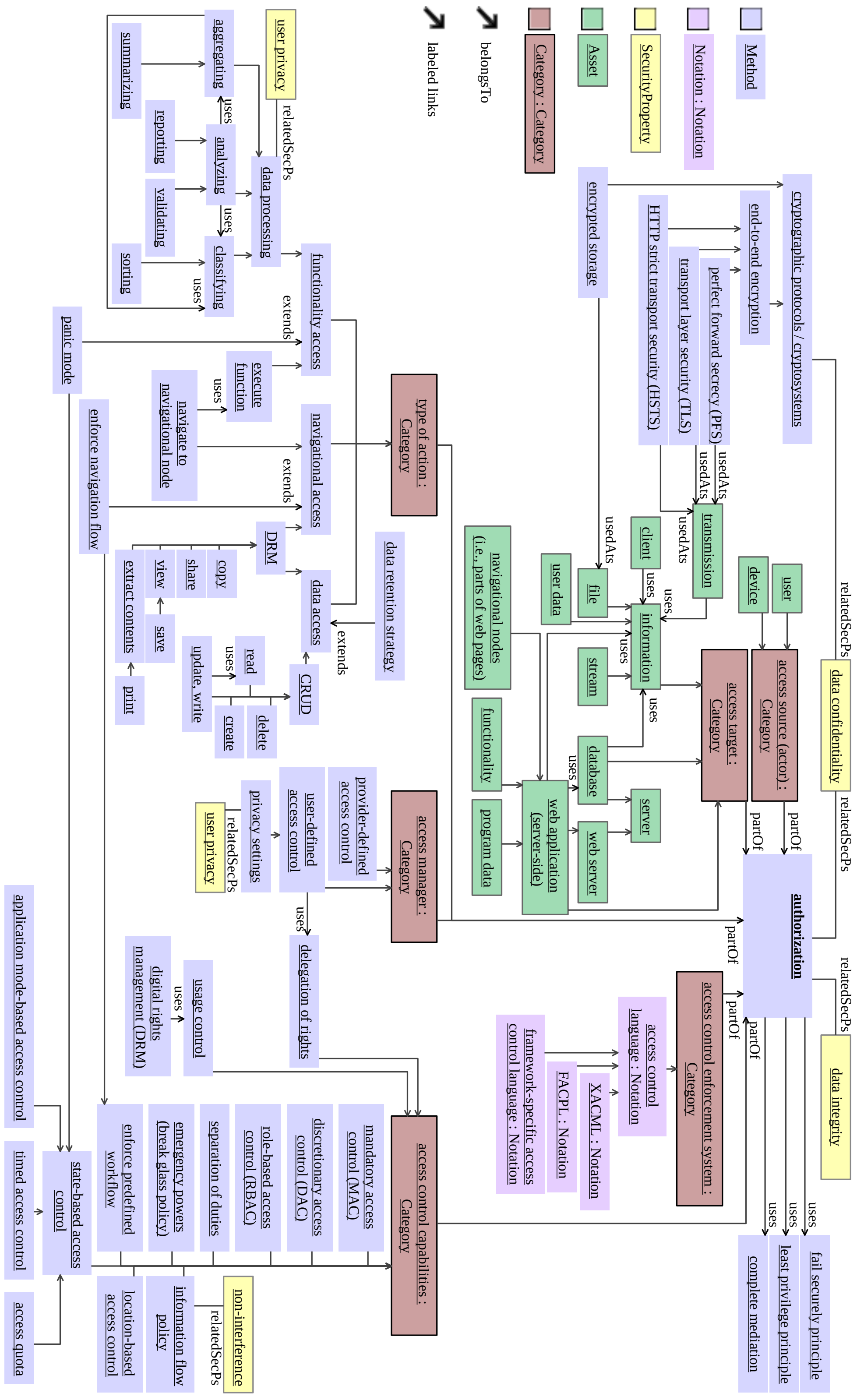

Method
Notation : Notation
SecurityProperty
Asset
Category : Category
belongsTo
labeled links
cryptographic protocols / cryptosystems
end-to-end encryption
perfect forward secrecy (PFS)
transport layer security (TLS)
HTTP strict transport security (HSTS)
encrypted storage
usedAts
transmission
client
file
user data
information
stream
database
server
web server
web application (server-side)
program data
functionality
navigational nodes (i.e., parts of web pages)
uses
user
device
access source (actor) : Category
access target : Category
partOf
relatedSecPs
data confidentiality
authorization
data integrity
access control enforcement system : Category
access control language : Notation
XACML : Notation
FACPL : Notation
framework-specific access control language : Notation
fail securely principle
least privilege principle
complete mediation
type of action : Category
access manager : Category
provider-defined access control
user-defined access control
privacy settings
user privacy
delegation of rights
access control capabilities : Category
mandatory access control (MAC)
discretionary access control (DAC)
role-based access control (RBAC)
separation of duties
emergency powers (break glass policy)
enforce predefined workflow
information flow policy
location-based access control
non-interference
state-based access control
application mode-based access control
timed access control
access quota
usage control
digital rights management (DRM)
data retention strategy
data access
extends
CRUD
read
delete
create
update, write
DRM
copy
share
view
save
extract contents
print
navigational access
navigate to navigational node
enforce navigation flow
functionality access
execute function
panic mode
data processing
classifying
analyzing
aggregating
summarizing
reporting
validating
sorting

Figure 4.7: SecWAO methods: authorization

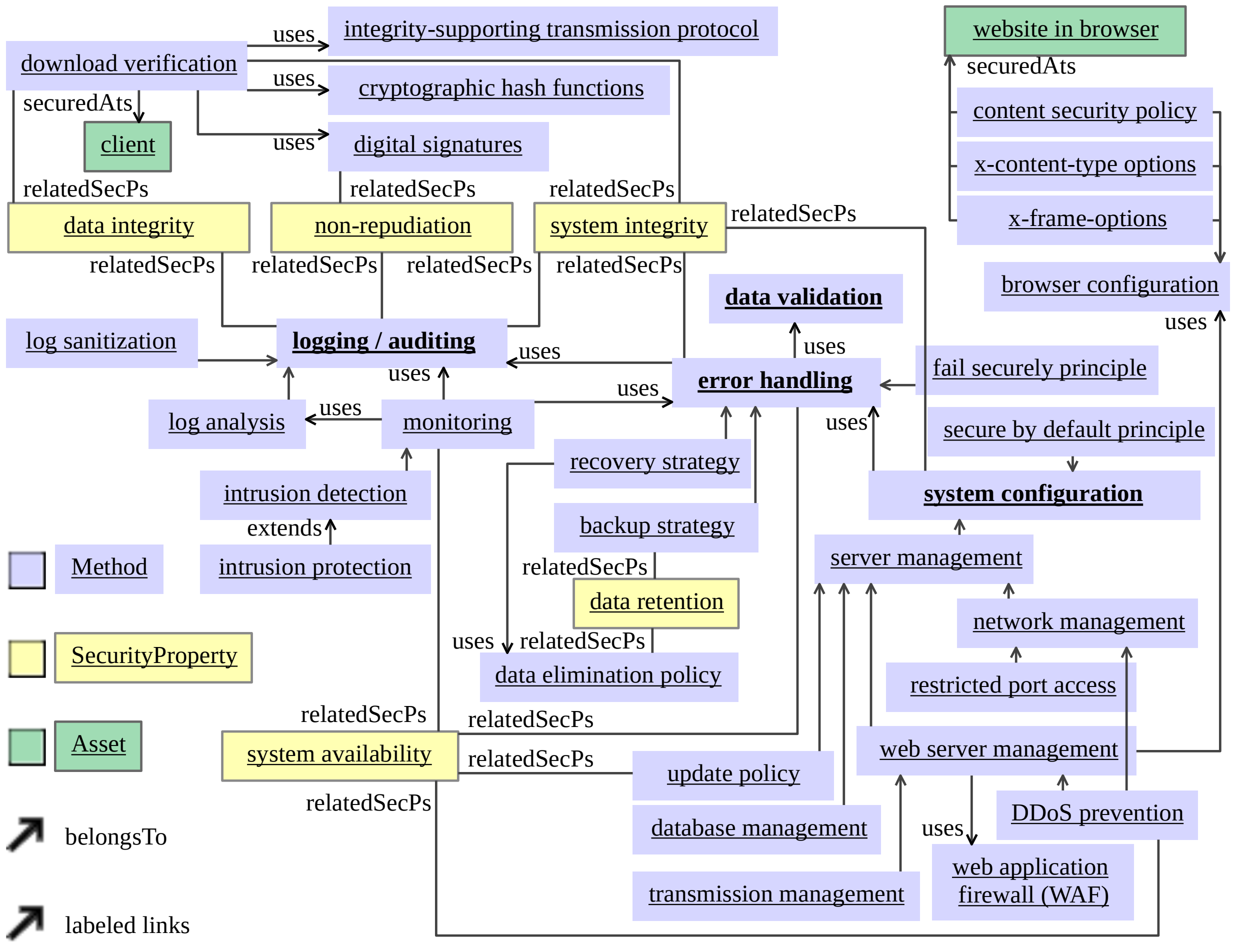


Figure 4.8: SecWAO methods: logging, error handling and system configuration

Monitoring uses logging and log analysis for intrusion detection as well as for intrusion prevention. Log sanitization means that sensitive information is removed[38] from log messages. This can be important for information that has to be deleted after a certain period of time due to legal regulations and for debugging by developers who are not allowed to access concrete data sets of the production system. For error handling, a data validation method should be applied to error messages in order to keep details of the system and the algorithm internal. Consequently, an internal error message should be replaced by a short message, which does not confuse normal users and does not allow attackers to gain any insights. Additionally, the instance `fail securely principle` reminds developers to carefully consider error states as regular part of their program, as web applications should be constantly available – meaning that restarts to recover a consistent state are undesirable.

Regarding system configuration, we provide some examples in figure 4.8, as server management and browser configuration. Browsers can be configured by users, but their behavior can also be influenced by web servers' responses, as already introduced above, e.g., by using a content security policy and other http headers [165].

In general, objects of SecWAO can be grouped according to different aspects. For example methods could also be grouped according to a certain security property like data integrity, to learn about methods that are related to it. Purists might as well query the tree that contains all UML elements.

4.4 Vulnerabilities and Threats

Figure 4.9 depicts the ten top vulnerabilities of web applications (according to OWASP [161]) and relates them with major threats that may exploit these vulnerabilities. The diagram is roughly grouped according to the main methods we used in the previous section. For example, the instance of the class `Vulnerability` that is named `unvalidated data` (in bold) corresponds to the method's instance `data validation`, depicted in figure 4.5. These correspondences can be modeled using associations from figure 3.4 (i.e. with the roles `detectedVs` or `shieldedVs`), as presented in figure 4.1. Beyond OWASP's Top 10, different views on general vulnerabilities are presented by the CWE [66], e.g., they group them according to reasons for weaknesses[39] or according to diverse development concepts[40].

Besides the vulnerabilities from the OWASP Top 10, the diagram in figure 4.9 depicts several related vulnerabilities ranging from general vulnerabilities like error-prone memory management or insecure credentials[41] to web-specific vulnerabilities like clickjacking[42] or Cross-Site

[38]Wikipedia: Sanitization (classified information)

[39]CWE Research Concepts. https://cwe.mitre.org/data/graphs/1000.html

[40]CWE Development Concepts. https://cwe.mitre.org/data/graphs/699.html

[41]Insecure credentials are, e.g., passwords that are easily guessable. This can mean that they are either too short so that a brute-force attack is possible, or too common so that a rainbow-table attack is effective. Both exploit that brute-force credential guessing is possible, either on stolen password hashes (in case they are not hashed with an up-to-date cryptographic hash function and salted) or on web applications that apply no means to restrict credential guessing attempts. (cf. lower right of figure 4.9)

[42]A web page is vulnerable to clickjacking if an attacker can hide it by layers with arbitrary contents to trick users into clicking on these layers and thus involuntarily interact with the hidden web page [163].

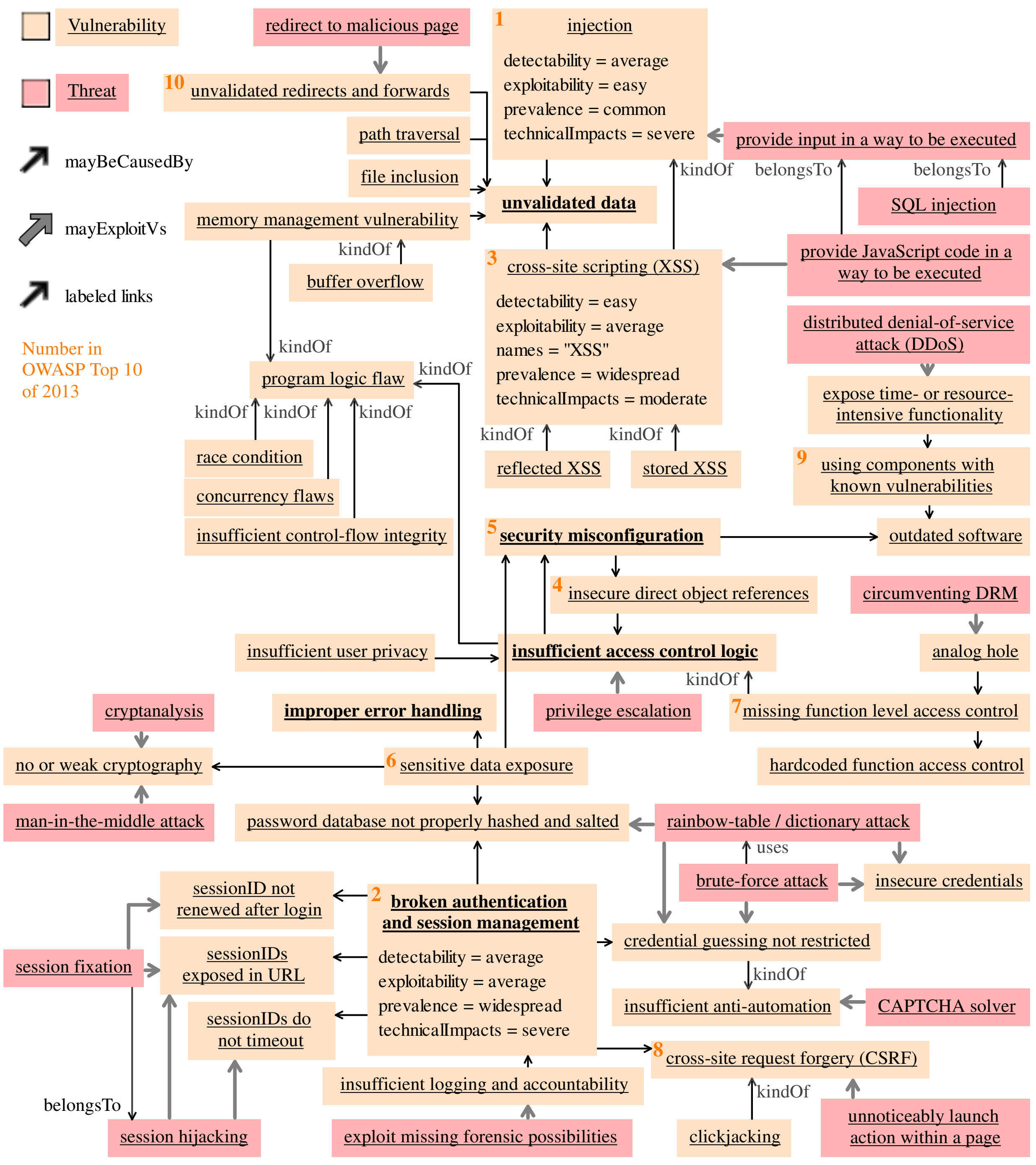

Vulnerability
Threat
mayBeCausedBy
mayExploitVs
labeled links
Number in OWASP Top 10 of 2013
redirect to malicious page
10 unvalidated redirects and forwards
path traversal
file inclusion
memory management vulnerability
buffer overflow
1 injection
detectability = average
exploitability = easy
prevalence = common
technicalImpacts = severe
provide input in a way to be executed
SQL injection
unvalidated data
provide JavaScript code in a way to be executed
3 cross-site scripting (XSS)
detectability = easy
exploitability = average
names = "XSS"
prevalence = widespread
technicalImpacts = moderate
reflected XSS
stored XSS
distributed denial-of-service attack (DDoS)
expose time- or resource-intensive functionality
9 using components with known vulnerabilities
outdated software
program logic flaw
race condition
concurrency flaws
insufficient control-flow integrity
5 security misconfiguration
4 insecure direct object references
insufficient user privacy
insufficient access control logic
circumventing DRM
analog hole
privilege escalation
7 missing function level access control
hardcoded function access control
cryptanalysis
improper error handling
no or weak cryptography
6 sensitive data exposure
man-in-the-middle attack
password database not properly hashed and salted
rainbow-table / dictionary attack
uses
brute-force attack
insecure credentials
sessionID not renewed after login
2 broken authentication and session management
detectability = average
exploitability = average
prevalence = widespread
technicalImpacts = severe
credential guessing not restricted
session fixation
sessionIDs exposed in URL
sessionIDs do not timeout
insufficient anti-automation
CAPTCHA solver
8 cross-site request forgery (CSRF)
insufficient logging and accountability
session hijacking
exploit missing forensic possibilities
clickjacking
unnoticeably launch action within a page
kindOf
belongsTo

Figure 4.9: SecWAO vulnerabilities

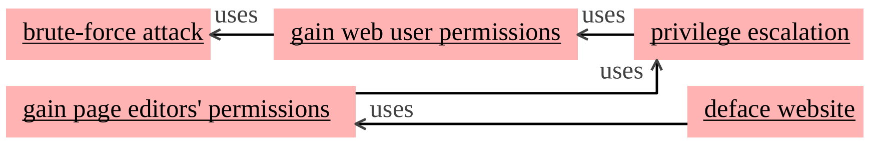


Figure 4.10: SecWAO: a sequence of threats

Scripting. Depending on the configuration of a concrete asset, some vulnerabilities, like `weak credentials`, can only be exploited in combination with others.

In practice, threats and attacks often share the same name, as e.g., the *threat* of a brute-force *attack*. In Herzog et al. [101, figure 4], a threat classification is provided that uses the terms "threat" and "attack" interchangeably. Fenz and Ekelhart [83] differentiate in their online ontology between "low level threats" and "top level threats". The former correspond to our definition of threats, the latter are the counterpart to what we express positively as security property (e.g., data disclosure – confidentiality). For us, attacks are threats that become reality for a concrete asset.

Successful attacks can give rise to further vulnerabilities, as an attacker can use a found out password to search for vulnerabilities that are not exploitable from the outside. As the class `Threat` descends from `Method`, the roles `uses`, `steps`, `extends`, etc., can be used. Figure 4.10 shows an example of a sequence of threats that could become reality so that in the end a set of web pages are defaced. In this way, attack trees [185][43] can be constructed. Such trees can grow exponentially[44], as threats depend on assets: storing illegal data might be a huge threat for servers with plenty of storage, whereas misconfiguring nuclear power plants might be less relevant for average server administrators.

A general collection of threat trees can be found in Shostack's book [197, appendix B], which follows Microsoft's STRIDE threat model[45]. The items of STRIDE stand for Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service and Elevation of Privilege.

In the CWE [66] and in CAPEC [212], consequences of a weakness respectively of an attack pattern can be chosen from: confidentiality, integrity, availability, non-repudiation, accountability, authentication, authorization, access control and "other". SecWAO distinguishes between security properties (the former) and methods (the latter), which makes it possible to provide detailed information about the location and the consequences of a vulnerability in a structured way. For example, an unknown conceptual vulnerability in a method might be implemented in a tool. If it is critical for a piece of software that this tool works like intended, it can be modeled as an asset itself (cf. roles `Vulnerability.includedIn` and `Asset.includes` in figure 3.4).

4.5 Implementation of a Knowledge Base

Using a web application can be more convenient than using UML CASE tools, especially when it comes to collaborative research. Advantages of a web-based implementation for managing

[43]Although the term "attack tree" is widely used, the term "threat tree" is more appropriate, as there is no need for attackers to exploit all theoretical possibilities.

[44]The growth of the ontology is a reason why SecWAO will never be completely finished; aside from the fact that the state of the art changes over time.

[45]The STRIDE Threat Model. https://msdn.microsoft.com/en-US/library/ee823878.aspx

SecEval's instance models are that connections to existing knowledge objects (e.g., common methods or vulnerabilities of SecWAO) can easily be added and that data sets of previous evaluations remain available for future research. In section 4.5.1, we elicit requirements for a Wiki-like knowledge base for software and security engineers and developers and in section 4.5.2 we present our prototypical implementation.

4.5.1 Requirements

We think that the SecEval Wiki should support the following use cases: (a) viewing knowledge objects, (b) editing knowledge objects, (c) importing external information and (d) searching for information to answer research questions, which can result in executing a tool-supported SecEval evaluation process.

Viewing Knowledge Objects

For the implementation of SecEval's Security Context model, we imagine a system that provides three views on each knowledge object:

- A *tabular view* that shows attributes' values, grouped by classes (presented as boxes) of SecEval's models. Which attributes are shown can be defined by the user. This view is especially useful for comparing knowledge objects.
- A *UML view* that presents an instance model of SecEval's UML model. The advantage of this view is that it is easy to examine links between several knowledge objects.
- A view that shows *continuous text*, enriched by boxes that can be placed between paragraphs or beside the text, similar to Wikipedia[46].

The user should be able to switch between these views at any time.

Editing Knowledge Objects

When creating a new element in the Wiki, the page is empty at first and shown in the *continuous text view* so that text can by written and structured by headings and paragraphs immediately, similar to the workflow in traditional Wikis. Additionally, on the side of the screen, common attributes are presented in a sidebar which can be dragged onto the Wiki page in order to fill them with actual values and to arrange them within the text or in boxes. These attributes correspond to attributes from SecEval models. For example, the user can specify some attributes of the `Tool` class, as technical requirements, licenses or the language the tool is written in.

Usually, information about knowledge objects has already been stored in continuous text form (probably imported from another page or document). In such cases, the application should allow to easily mark text and to click on an attribute on the sidebar. The attribute is then linked to the text so that it changes automatically when the text is altered. A history not only for continuous text, but also for attributes, allows users to track how the contents evolved over time.

For the sidebar (which should be resizable up to full-screen) it is also useful to implement different views, for example:

[46]Wikipedia. https://www.wikipedia.org/

- a *UML view*, showing the full SecEval class diagrams for experts. This is the counterpart to the instance view for a concrete entry of the Wiki.
- an *auto-suggestion view* in which single attributes are shown according to an attribute-based suggestion system. This system can then recommend attributes which seem to be useful in the current context, as e.g., attributes of testing tools, as soon as it becomes clear that a user describes a tool from the domain of testing.

Recommendation includes that the system needs to explain rules inferred by SecEval, as e.g. that it is useful to describe a tool and a corresponding notation in two separate entries, even if the notation has only been used by this tool so far. A focus is on the connection between several knowledge objects in our SecEval system and on the possibility to add data which is not only associated with one knowledge object, as e.g., evaluation results.

Importing External Information

Another useful feature is syndication, i.e. to be able to insert text from other web pages, as from Wikipedia or from vulnerability management systems, which are correctly cited and updated automatically. This task could be eased by step-by-step wizards and good attribute recommendation according to the attributes selected so far and the information provided. For example, if the user inserts the URL of a Wikipedia article, the article is displayed in a window that allows selecting passages and transferring them to the SecEval system immediately, along with a linked cite.

Another requirement is the import of text from pdf files. Hereby, a challenge is to deal with licensed books or papers, because citing small passages is usually allowed, whereas publishing the whole document in the web might be prohibited.

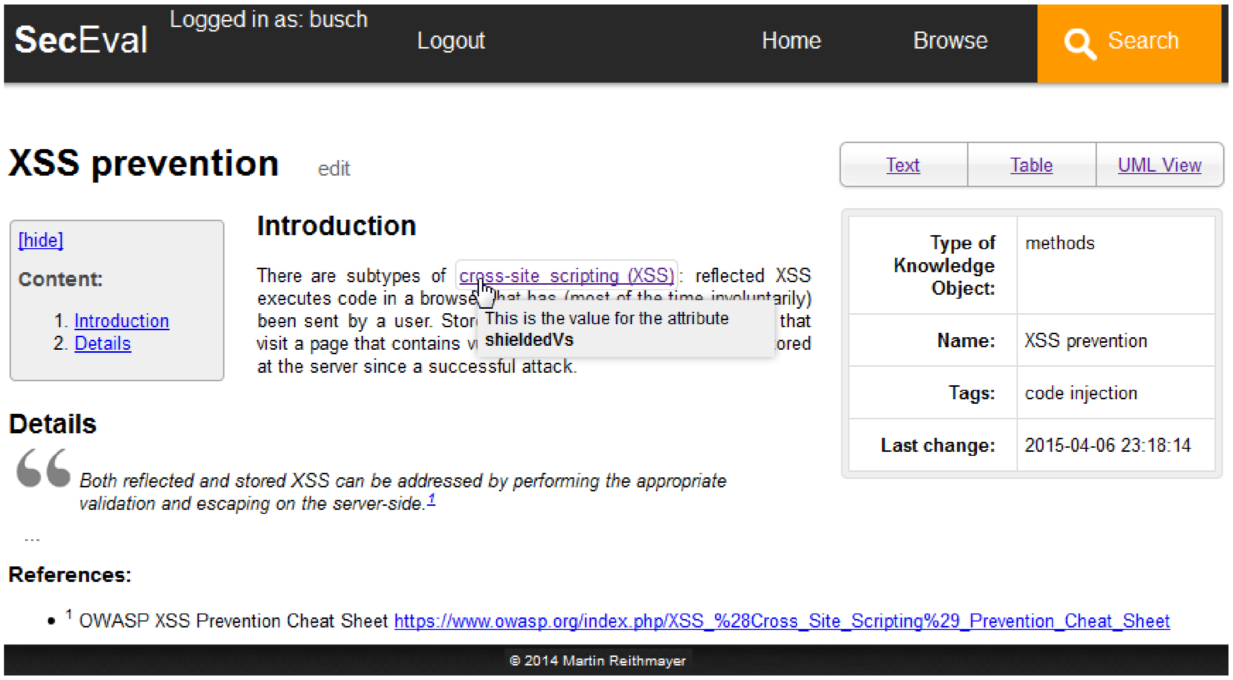
Searching for Information

In addition to the implementation of the Security Context model, to express elements of SecWAO or gathered information from evaluations, the application should support the process of collecting and analyzing data to answer a concrete research question.

Simple questions can be answered using a full-text search. More complex questions can involve several knowledge objects and their attributes, so that the search function has to be able to rely on the associations between knowledge objects stored in the knowledge base.

If the requested information cannot be found in the knowledge base, a wizard might suggest using SecEval's process to collect and analyze information. Ideally, the wizard allows jumping between several process steps while offering to record information for SecEval's Data Collection and Data Analysis models. The users can decide whether their research question should be public.[47] At the end of a complex evaluation process, artifacts like research questions, used sources and the concrete approach of a research can be published to save time and money in case a similar question will arise again in the future. Documenting the process of data collection and data analysis could avoid misconceptions regarding the validity and reusability of results.

[47]Discussions can also help to answer a research question, therefore it is desirable to connect the Wiki with question/answer systems as, e.g., Stackoverflow http://stackoverflow.com/.


SecEval Logged in as: busch Logout Home Browse Search

XSS prevention edit

Text | Table | UML View

[hide]

Content:

1. Introduction
2. Details

Introduction

There are subtypes of cross-site scripting (XSS): reflected XSS executes code in a browse[r] that has (most of the time involuntarily) been sent by a user. Stor[...] that visit a page that contains v[...]ored at the server since a successful attack.

This is the value for the attribute **shieldedVs**

Details

Both reflected and stored XSS can be addressed by performing the appropriate validation and escaping on the server-side.[1]

...

References:

- [1] OWASP XSS Prevention Cheat Sheet https://www.owasp.org/index.php/XSS_%28Cross_Site_Scripting%29_Prevention_Cheat_Sheet

Type of Knowledge Object:	methods
Name:	XSS prevention
Tags:	code injection
Last change:	2015-04-06 23:18:14

© 2014 Martin Reithmayer

Figure 4.11: SecEval implementation: continuous text view

A general requirement for our implementation is the usability of the interface. For example, the CBK provides a complex search function, but it turned out that it is rarely used, because attributes have to be selected by using their technical short names. For SecEval, it might be helpful to present descriptions and to suggest attributes according to a catalogue that learns how users tend to name a concept. Ideally, this search does not require a complex interface, but supports the user with auto-completion or wizards when typing a query into a text box.

4.5.2 Implementation

In section 2.2.2, we introduced existing knowledge base systems. However, they are not flexible enough to meet our requirements, especially when it comes to editing knowledge objects' information without being restricted by a rigid structure (as e.g., enforced by the CBK's Semantic MediaWiki [120]) or to usability (e.g., drag & drop for inserting referencing tags to knowledge object's attributes in a wiki page). Therefore, we developed a prototype of an online knowledge base, implemented by Martin Reithmayer [176], which is based on the requirements described in the previous section.

Section 4.5.2 depicts a screenshot of the continuous text view for the method "XSS prevention" from our SecWAO. The views can be changed in the upper right corner. The introduction references the vulnerability "Cross-Site Scripting (XSS)" that should be shielded by this method (cf. UML view, which is depicted in section 4.5.2). The link represents the value of the method's attribute `shieldedVs`. In the edit view, an "attribute picker" allows to select and search for attributes that can be added to the text using drag & drop, which inserts a reference tag in wiki syntax, in this case `[[attribute:t:shieldedVs]]`.

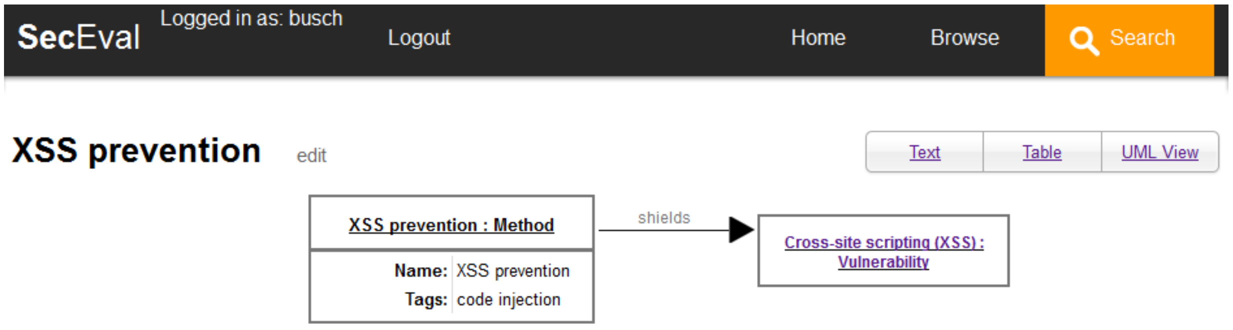


Figure 4.12: SecEval implementation: UML view

In the "Details" section, information from the OWASP website is cited, which was selected using the implemented HTML import functionality of the SecEval Wiki. The import dialogue shows a preview of a website and allows the user to select an excerpt of the text. This excerpt can then be included in a wiki article using tags, which automatically adds a reference, as shown at the bottom of section 4.5.2.

Editing attributes is depicted in section 4.5.2: users can drag & drop attributes from a list on the right to the left, where users can then specify a value for these attributes. If the view is changed to "text fields", the general wiki page (continuous text view) can be edited using these values, as described above.

Technically, the SecEval Wiki was implemented using PHP [173] and a MySQL database [160] for the back-end and HTML, CSS and JavaScript with the JQuery library [108] for the front-end. More detailed information about the implementation can be found in [176].

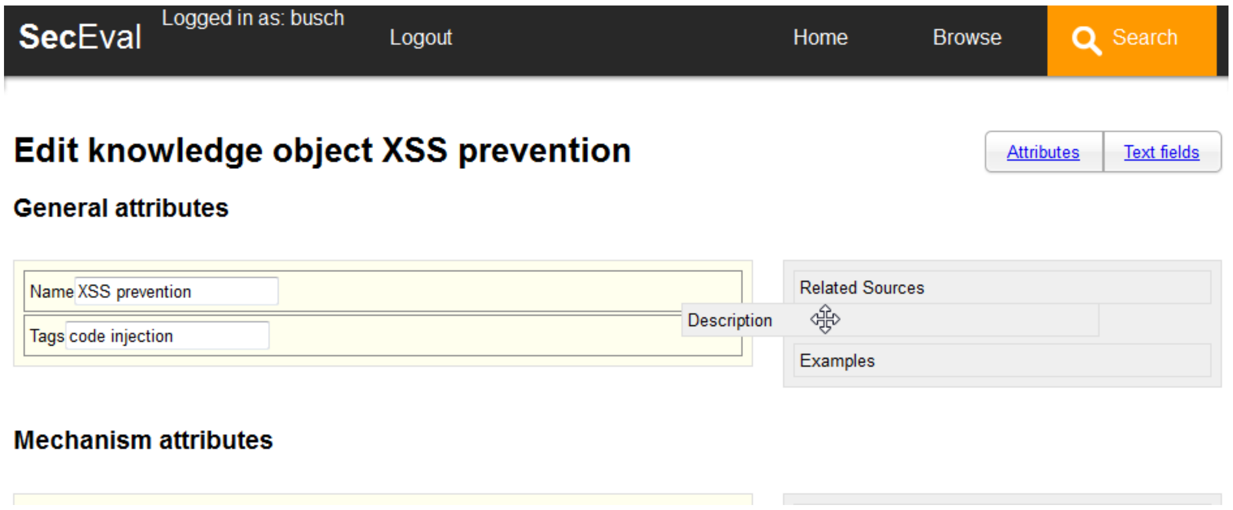


Figure 4.13: SecEval implementation: edit attributes

4.6 Summary and Related Work

In this chapter, we answered our second research question (RQ2)[48] by presenting our Secure Web Applications' Ontology (SecWAO), which is based on SecEval's Security Context model. Its main goal is to raise the awareness for web security concepts and especially for their interconnections.

SecWAO. SecWAO was built as an UML instance model of SecEval's Security Context model. No extension of SecEval was needed for modeling complex relations between central web security concepts, which additionally validated the expressiveness of the Security Context model. For example, SecWAO models security-related methods like cryptography, data validation, authentication, session management, authorization, logging, error handling, and system configuration. Instances of associations links them to other knowledge objects, as e.g., to security properties like confidentiality or vulnerabilities like XSS.

To the best of our knowledge, SecWAO is the first ontology that provides a comprehensive view of the domain of secure web applications. Main approaches we compared SecWAO with are: the CWE [66], Kim et al. [112], Herzog et al. [101] and Fenz and Ekelhart [83] (cf. section 2.2.2). For example, in the extensive CWE [66] and in CAPEC [212], consequences of a weakness respectively of an attack pattern can be chosen from: confidentiality, integrity, availability, non-repudiation, accountability, authentication, authorization, access control and "other". SecWAO distinguishes between security properties (the former) and methods (the latter), which makes it possible to provide detailed information about the location and the consequences of a vulnerability in a structured way.

Implementation. In addition to the representation of SecEval instances as UML diagrams, we implemented a prototype of a SecEval knowledge base, which is both flexible and simple to use. Familiarity with CBK [49], which is based on the Semantic MediaWiki [120], was helpful for figuring out key requirements. In particular, our SecEval Wiki allows viewing knowledge objects (text view, table view or UML class view) and creating or editing knowledge objects (its text or attributes). When editing the text view, information from the web can be selected for citation so that existing pieces of information can be organized with SecEval's Security Context model. The focus was on structuring information without limiting the user to the underlying structure, as enforcing a structure turned out to be an impediment when working with the CBK's implementation.

Outlook. In [144], Neuhaus et al. state that the first question to be asked when dealing with an ontology is: "Can humans understand the ontology correctly?". According to our experience, SecWAO reaches this goal due to its clear structure and its common terminology. So far, we have used SecWAO in two different ways: First, we structured a tutorial about practical IT security for master students according to SecWAO in the winter terms 2014/2015 and 2015/2016. Zooming into SecWAO UML diagrams on the slides allowed giving further explanations without losing track of the context. The students acknowledged the helpfulness of SecWAO for studying,

[48] RQ2 (extended): How are assets, security engineering methods, notations, tools, security properties, vulnerabilities and threats related in the domain of web application security? Is SecEval's Security Context model powerful enough to express these relations? (cf. section 1.2 and introduction of chapter 4)

although they noted that readability naturally depends on a projector with high resolution and a large projection surface.

Second, we integrated SecWAO into the UML-based Web Engineering (UWE) approach [224], as will be described in part III. The UWE notation is defined using the UML profile mechanism and SecWAO allowed us to systematically establish UML stereotypes and tags for modeling secure web applications. As a result, UWE enables developers to document most important security design decisions in a graphical way. We expect that this structured and concise documentation facilitates the development and maintenance of web applications, in particular in the case of changes in the developer team. Consequently, we can affirm our initial question about the power of SecEval's Security Context model as a fundamental structure for SecWAO.

Regarding our implementation of the SecEval Wiki, we think future work could extend it by a wizard that supports SecEval's process of collecting and analyzing information, an import from document file formats like pdf and an attribute-based suggestion system. Furthermore, a more detailed search engine and a version control system could be integrated.

Part III

Engineering Secure Web Applications

Many general lists of security requirements for web applications exist, e.g. [162, 187]. These abstract requirements often remain rather vague until being enriched by design instructions, as "If TLS is used, all components of the web application must be transmitted in encrypted form" [95, Req 3.06-2]).

Ideally, these statements are refined for concrete (web) applications: Both, developers and administrators have to know which methods, as e.g. TLS, are (going to be) employed to secure their web applications and how their applications are structured. This is where our approach bridges the gap from general requirements (e.g., "sensitive data is always transmitted confidentially") or general design decisions ("TLS is used") via concrete requirements of web applications ("users submit their confidential credit card number to the payment application") through to design decisions for these applications ("TLS is deployed for the payment service, but not for browsing offers").

Making concrete design decisions is closely related to choosing a method in the sense of a SecEval method, which is instantiated by SecWAO, e.g., as "Transport Layer Security (TLS)" (cf. part II: figure 4.7). Our approach, which extends a notation called UML-based Web Engineering (UWE) to answer RQ3, can be used to model applications according to their web-specific characteristics and to document security-related design decisions directly in these models.

Part III consists of two main chapters:

Modeling Secure Web Applications. UWE, which was invented by Koch et al. [115, 116], allows for modeling web applications graphically by using different views, such as content or navigation. We start with an introduction of the UWE approach along with several case studies we used for validating the soundness and applicability of our approach while developing it. We continue with the presentation of our UWE security extensions, which we structured by SecWAO's main security requirements and methods. In 2010, the author already started to add some security features (in the course of her master's thesis [37, 39], published 2011) which led to a number of security extensions over the years.

Artifact Generation. We explore artifacts that can be generated from UWE models: We sketch TextualUWE in section 6.1, which is a Domain-Specific Language (DSL) for expressing UWE models in a textual way. A transformation to convert graphical models in textual ones was partly implemented in a bachelor's thesis [181] under the supervision of the author. Section 6.2 highlights how to export access control policies in two different formats, namely XACML and FACPL, which is joint work with the inventors of FACPL [43]. In section 6.3 we report on work in which we joined forces with researchers from the domain of policy testing to build a toolchain that automatically tests generated XACML policies [20]. In section 6.4 we briefly present the idea of a transformation from UWE to ActionGUI models, called ActionUWE [38]. ActionUWE is joint work with the inventors of ActionGUI, a modeling method for secure web applications that allows generating executable code for models, which are limited to predefined elements. In section 6.5, we present an approach for testing and enforcing secure navigation paths, which is also joint work [47]. Navigation paths are the paths a user should stay on while browsing in order to comply with the intended application logic.

The interested reader can download the UML profile for UWE, our case studies and related tools from the UWE website [224].

Chapter 5

Modeling Secure Web Applications

In order to pursue the first objective of RQ3 – which is to examine how security aspects of web applications can be expressed – we need to choose a modeling language that is capable of representing the architecture of web applications, as well as their requirements and design. We selected the modeling language UML-based Web Engineering (UWE) [115, 116], as discussed in section 2.3.2. UWE is flexible due to its views, which are represented as UML models, and extensible due to UML's profiling mechanism.

Technically, UWE is a UML profile, which contains a set of stereotypes, tag definitions and patterns for modeling web applications with UML. Stereotypes can be added to a certain type of UML elements and they are displayed as ≪stereotypeName≫ or as icon. Stereotype can have typed tags. If an element is stereotyped, concrete values for tags of this stereotype can be set, which is denoted by a set of {tag=value} entries. UWE's UML profile can be downloaded from the UWE website [224]. Excerpts of the UWE profile are depicted in appendix C. Note that UWE allows to model the web application itself. Thus, is not designed to model client-specific security measures, as e.g., up-to-date browsers that users should have installed.

This chapter briefly introduces our case studies and main UWE models, before discussing how security modeling can be integrated into UWE, i.e., be expressed with the Unified Modeling Language.

5.1 Overview of Case Studies

Having a number of case studies was helpful for developing and evaluating our modeling approach. In this chapter, we briefly introduce our case studies and emphasize interesting requirements that are revisited and modeled in the remainder of this thesis. All UWE models of our case studies can be found on the UWE website [224].

In the area of SmartGrid, we modeled three scenarios: (1) the Energy Management System (EMS), which is an appliance that manages energy in a household. We published this case study and its detailed security requirements in [44] and an extended description can be found in appendix D. (2) An application that sells normal and special offers, which we published in [20] and (3) a SmartGrid Bonus application that can show a bonus code after an offer is bought, which we published in [47]. The Energy Management System (EMS) [64] as well as a patient monitoring system [63][1] were modeled in the scope of the EU project NESSoS. Both are based on requirements from Siemens. In addition, a model from a Hospital Information System[2] that was created during the author's master's thesis [37] was reused as a case study for artifact generation (cf. chapter 6). Besides, Fritsch [87], a master student supervised by the author, modeled the open-source web application ownCloud [157][3]. The following brief descriptions and figures are adapted from the aforementioned publications.

5.1.1 SmartGrid

The SmartGrid case study is split into three projects: an Energy Management System (EMS), an application that sells normal and special offers and a SmartGrid Bonus application that can show a bonus code after an offer is bought.

Energy Management System (EMS). The EMS is an interface for the SmartGrid customer that displays consumption data. Concrete instantiations can be realized by a web application that provides functionality for energy trading or for regulating the current drain. Ideally, most appliances, as e.g., ovens, dishwashers, washing machines or lamps are so-called Smart Appliances (SAs), which means they contain a small embedded-system that receives control commands from the EMS and that informs the EMS about the current status. Additionally, SAs can be controlled by pushing a button or by using an integrated touch screen.

For a household, exactly one EMS and one Smart Meter are installed locally, in a place where they are protected from physical tampering. The Smart Meter is responsible for monitoring the amount of energy that is sold or bought. As the EMS is connected to the web, remote access to its web application allows users to interact with the EMS and to monitor energy consumption from outside their homes.

Requirements for the web application of the EMS are that a user can buy or sell energy, control local energy consumption by configuring SAs, install plugins to automate tasks or manage other users. Configuring an SA over the EMS could, e.g., mean to limit the allowed usage time of a SmartTV by children. Our focus in this case study is primarily on security mechanisms, like authentication, panic mode, reauthentication, secure connections, authorization, user zone concept, cross-site-request-forgery prevention, under attack mode and SQL-injection prevention. An extended description of our EMS case study and its UWE models can be found in appendix D.

SmartGrid Offers. Requirements for the SmartGrid Offers application are that offers can be bought from a list of offers, which is generated individually for each user. Offers are connected to energy transactions, i.e. an amount of energy that is provided. Two types of offers exist: normal offers and special offers. Special offers are promoted, which means they are advertised at the beginning of the list of offers. Special offers can only be created by commercial users that provide a great quantity of energy, as owners of power stations do. Normal offers can be submitted by commercial users as well as by private households, which e.g., want to sell surplus energy from their solar panels. We assume that a concrete user can play one or both roles (private user or commercial user) at the same time. The focus in this scenario is on access control.

[1]Patient Monitoring case study. http://uwe.pst.ifi.lmu.de/examplePatientMonitoring.html
[2]HospInfo. A secure hospital information system. http://uwe.pst.ifi.lmu.de/exampleHospInfo.html
[3]OwnCloud case study. http://uwe.pst.ifi.lmu.de/exampleownCloud.html

SmartGrid Bonus. Basically, our SmartGrid Bonus application represents a prototype of an energy offer management including optional bonus handling. It provides two different user roles namely providers and customers: Providers manage and sell energy packages including optional bonus programs for customers. Customers have the possibility to buy offered energy packages. Therefore, our application lists all available energy offers and the customer selects a specific offer which includes a bonus code. After buying an energy package, the application shows the corresponding bonus code which contains a gift voucher, e.g., for online shops. Finally, the customer gets a confirmation for the ordered energy. In this scenario, we focus on the sequence of events, which corresponds to navigation possibilities in the web applications that are dynamically locked or unlocked.

5.1.2 Patient Monitoring

Patient monitoring aims to collect health-related data independently of the patient's location. This helps not only to sample data under every-day conditions, at work or at home, but also gives control to the patients, because they can get immediate feedback in critical situations. For instance, a wrist watch with a wearable sensor (referred to as "wearable" or simply as "(mobile) device") can measure vital signs as blood pressure or heart rate and physicians can add context-related advices and alarms to guide their patients.

In an associated web application, both, physicians and patients can configure uncritical alerts, as e.g., a ring tone for taking a medicine. Only physicians can configure critical alerts and asking patients for consent (e.g., ask if their data from the wearable can be used for scientific studies). When giving a wearable to a patient, ensuring correct authorization is very important, as otherwise data of one patient might show up in the health records of another patient, someone might eavesdrop the monitored data or manipulate the wearable in a way that critical alarms are not shown.

A bootstrap procedure binds a public key and an eHealth system to the wearable. Afterwards, we assume it knows the IP address (or web address) of the eHealth back-end server, and the server knows the public key of the mobile device. For assigning a wearable to a patient, two methods are available: a main variant in which patients authenticate themselves in the web application and enter the ID of their device, wait until they receive a nonce both on their browser and on their wearable, and press ok in the web application and on the wearable. A second variant is using OAuth[4]. This case study focuses on early design activities regarding access control and navigation within the web application that is used by patients and physicians.

5.1.3 Hospital Information System

The case study is a prototype of a web-based Hospital Information System, called HospInfo, which manages patients, wards and users that can play several roles. Stored data is, e.g., a patient's name, birth year, gender, address and blood type. The roles identified for this web application are: visitor, registered user, nurse, receptionist, physician, and admin. Its main requirements are:

- staff members should be able to register
- an administrator can set roles to staff members

[4]OAuth 2.0 http://oauth.net/2/

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

As HospInfo was our first UWE case study with a focus on security, it was used to elicit basic security modeling requirements that could not be directly expressed with UWE before.

5.1.4 OwnCloud

OwnCloud is an open-source software that is designed to permit end users to run their own file hosting services. It consists of the ownCloud Server (OCS), which is the file hosting server residing on a network-accessible host, and clients for a number of different platforms, both mobile and desktop. The server, at its core, can be considered an alternative to the popular file hosting service Dropbox[5]: it allows users to store, download and upload files to and from a server. However, its functionality can be expanded by installing additional applications, e.g., a calender, task manager or an address book.

The back-end is written in PHP and uses a SQL database to store internal data. The core logic of the ownCloud server is written by ownCloud developers, but common functionalities like routing, dependency injection, database abstraction, etc, are handled by existing frameworks like Symfony[6] and Zend[7]. The web-accessible front-end is written in HTML and JavaScript and makes use of the jQuery framework[8] and a number of plugins that rely on jQuery.

It is worth mentioning that due to its modular and extensible nature, we focused on modeling its core functionalities, trying to depict what a typical instance could look like. At the beginning of the source code analysis the most recent version of ownCloud Server was 7.0.4. The main purpose of this case study was to validate that all security features of an advanced web application can be modeled with UWE, which was the case, although it became obvious that UML modeling in general reaches its limits when highly dynamic processes are involved at runtime.

5.2 UWE Models

This section briefly presents the modeling language UML-based Web Engineering (UWE). The main characteristics of UWE are the different views that are represented as UML models and the use of a set of stereotypes and tags, which define UWE's model elements in its UML profile.

There are mainly two different aims that could be pursued by modeling software: modeling for code generation and modeling for structuring, discussing and documenting thoughts about requirements or software architecture/design. In the first case, models have to be as detailed as code, which can lead to very large and complex models. Advantages are that model checking

[5]Dropbox. http://dropbox.com/
[6]Symfony. https://symfony.com/
[7]Zend. http://framework.zend.com/
[8]jQuery. https://jquery.com/

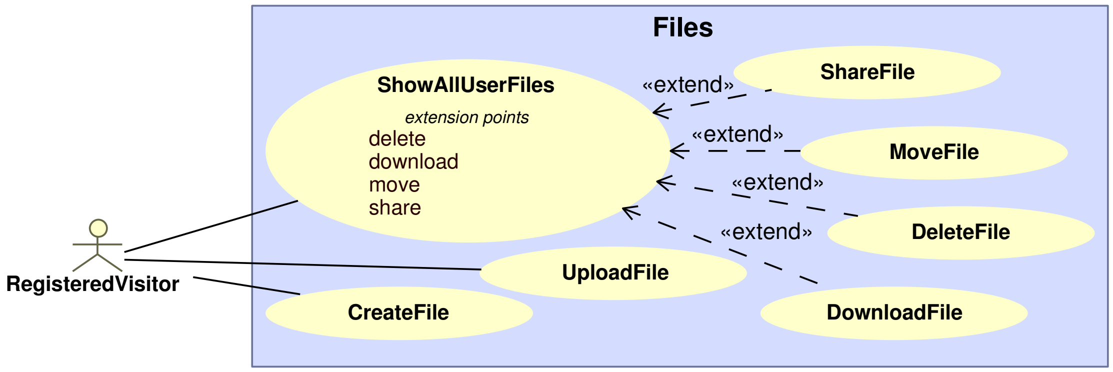


Figure 5.1: Example: Requirements (OwnCloud File Management)

might be easier than code analysis and that it might be better to generate code if the target language often requires detailed, but recurring instructions, as e.g., for memory management. In the second case, generating code often takes a backseat, in favor of a rather high level of abstraction. In this case, code can only be generated from more elaborated modeling parts or by using semi-automatic code generation, i.e., further information has to be provided during or after the generation process.

In our work, we mainly pursue the second goal and at the same time select (mainly security-related) aspects that are modeled in further detail, as e.g., access control. We recommend this process especially for web applications, as they tend to change a lot due to new technologies.

In the following, the UWE Requirements model, Content model, the Role model and the Basic Rights model, Navigation model, the Presentation model, and the Application States model are presented. Models can use different UML diagram types. UWE mainly employs use case diagrams, class diagrams and state charts. However, any other UML diagram type can be used additionally. For example, activity diagrams can be used for process modeling. They even were enriched by UWE stereotypes, e.g., in [119], but they are not in the focus of this thesis. However, we agree with OpenSAMM, which suggests developing "data-flow diagrams for sensitive resources" [52, p. 61], as for other resources the overhead is often too high.

It turned out to be useful to create a UML model for every kind of model in UWE. A model can contain several UML elements, as e.g., UML classes, associations or diagrams similar to a folder in an operating system. Models can have arbitrary names, which makes it difficult for algorithms to recognize the type of a model. That is why we introduced stereotypes for all UWE models, similar to the UML stereotype ≪useCaseModel≫ (which we use for UWE's Requirements Model, but ≪requirementsModel≫ is also defined in UWE): ≪contentModel≫, ≪roleModel≫, ≪basicRightsModel≫, ≪navigationStatesModel≫, ≪presentationModel≫, and ≪appModeModel≫.

5.2.1 Requirements Model

The Requirements model (sometimes also called use case model) defines functionality for an application and actors that use it at a high level of abstraction. This can be done by using plain UML use case diagrams, as depicted in figure 5.1, which shows use cases for the OwnCloud File Management system.

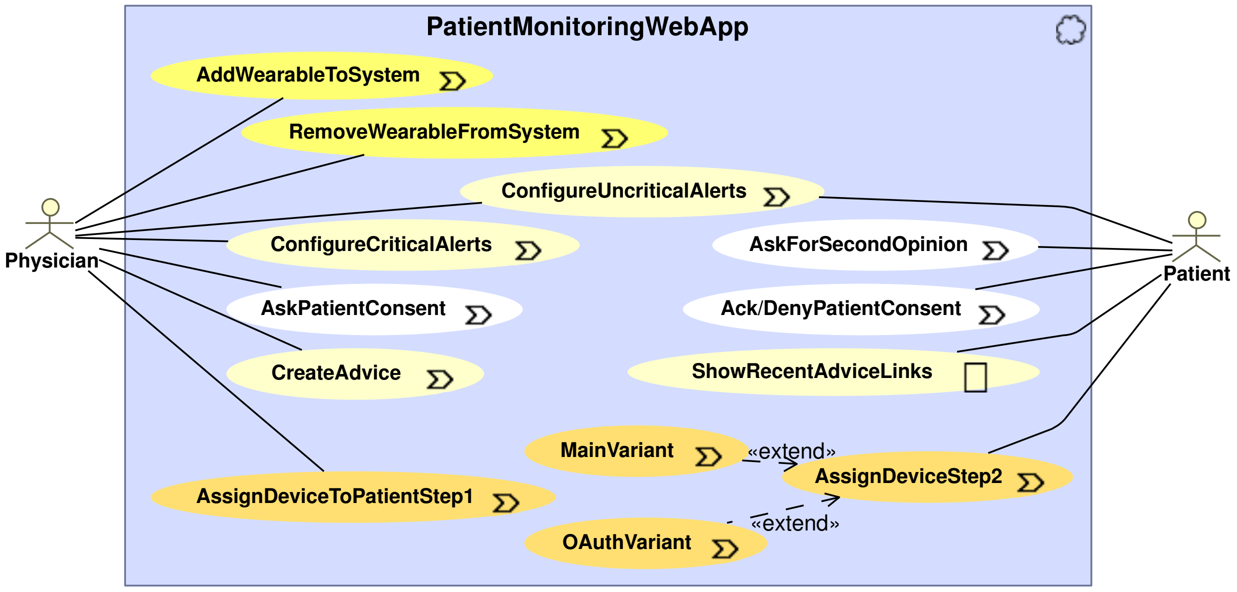


Figure 5.2: Example: Requirements (Patient Monitoring)

Additionally, UWE enriches UML's use case diagram with stereotypes, e.g., the ≪webUseCase≫ (◌) stereotype that can be applied on UML packages and use cases. Its {guard} tag allows to define high-level expressions that have to be true in order to be able to perform contained actions. The stereotypes ≪browsing≫ (□) and ≪processing≫ (⊳) are descendants of ≪webUseCase≫. The stereotype ≪browsing≫ can be applied on UML use cases to express that a use case only contains "browsing" activities, i.e. queries are executed that do not change application data. For example, figure 5.2 depicts[9] that showing advices from a physician does not change the application data. The stereotype ≪processing≫ expresses that, more often than not, application data is created or changed while the use case is processed. UML use cases can be grouped by packages. We define the shortcut that whenever a UWE stereotype is applicable to a package-like structure as well as to its containing elements, it only has to be applied to the package to denote that all compatible containing elements are stereotyped equally. More about requirements elicitation with UWE (without security concerns) can be found in the work from Koch and Kozuruba [117]. They, e.g., annotate UML activity diagrams so that common web-related requirements like periodic data refresh can be expressed.

Regarding security, threats can be modeled by misuse cases according to Sindre and Opdahl [198]. Although modeling activities that should not be successful contradicts the idea of the positive security model – which only defines what should be allowed – they can be helpful to draw attention to major risks.

5.2.2 Content Model

The Content model contains the data structure used by the application. "Data structure" usually refers to the structure of important classes in an object-oriented application or to the logical

[9]Colors in UWE diagrams are just used for grouping similar items or to increase readability

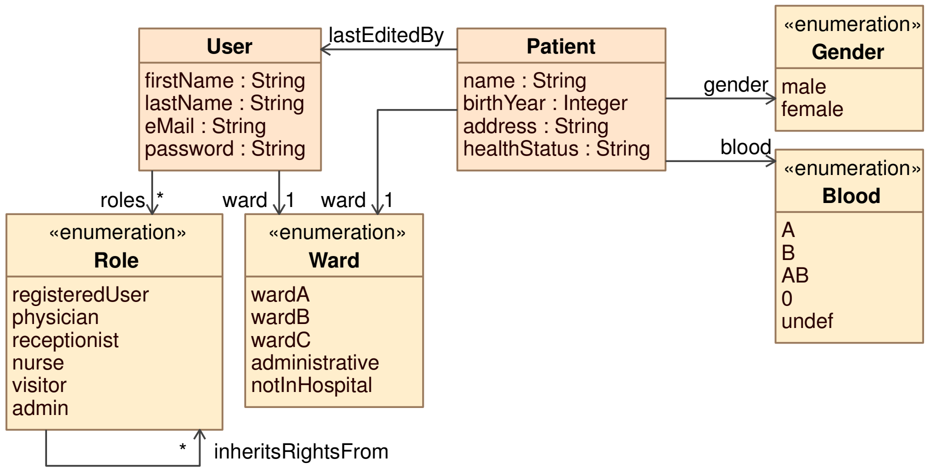


Figure 5.3: Example: Content (Hospital Information System)

structure of data in a database and in many times both are closely related. For non-object-oriented languages, Content classes can represent main modules or code files and their relations to each other, in case this turns out to be helpful for planning the structure of the application or for documentation purposes.

Class attributes or classes themselves can represent types of assets, as e.g., data specific to the individual. E.g., the patient's health status is an asset that can be managed by an hospital information system (figure 5.3). The Content model is used to have no UWE stereotypes, which will be changed in section 5.3 in order to be able to define security properties and used methods for these assets.

5.2.3 User Model, Role Model and Basic Rights Model

With the Role model, a hierarchy of user groups (roles) can be described. It is part of a User model («userModel») that allows modelers to create default users that should exist when an application is deployed, or to model example scenarios. UWE's User model defines the stereotype «webUser», which can be used on a UML class that should stand for the application's user. In the Role model, roles can be modeled as instances (or less frequently as descendants) of the class stereotyped by «webUser». Anonymous users can be addressed by using a role instance with a name like `anonymous:Role`.

The Basic Rights model describes access control policies. It constrains elements from the *Content model* by defining the access rights that should be granted to roles (or default users or other subjects from the *Role model*). Figure 5.4 depicts an example from our SmartGrid EMS case study. In the EMS system, a user can update or delete an instance of `User`, as long as it is not the current user. This is expressed by a UML comment that is stereotyped by «authorizationConstraint» and that contains the expression `pre: self <> caller`. "Self" refers to an instance of the constrained element, like the `User` class, whereas "caller" refers to the instance of the subject that wants to execute an action. "Pre:" can be used to stress that the expression has to be true before an action is executed. Note that figure 5.4 depicts default users. General access control rules for roles in the EMS are modeled in figure D.10 in appendix D.

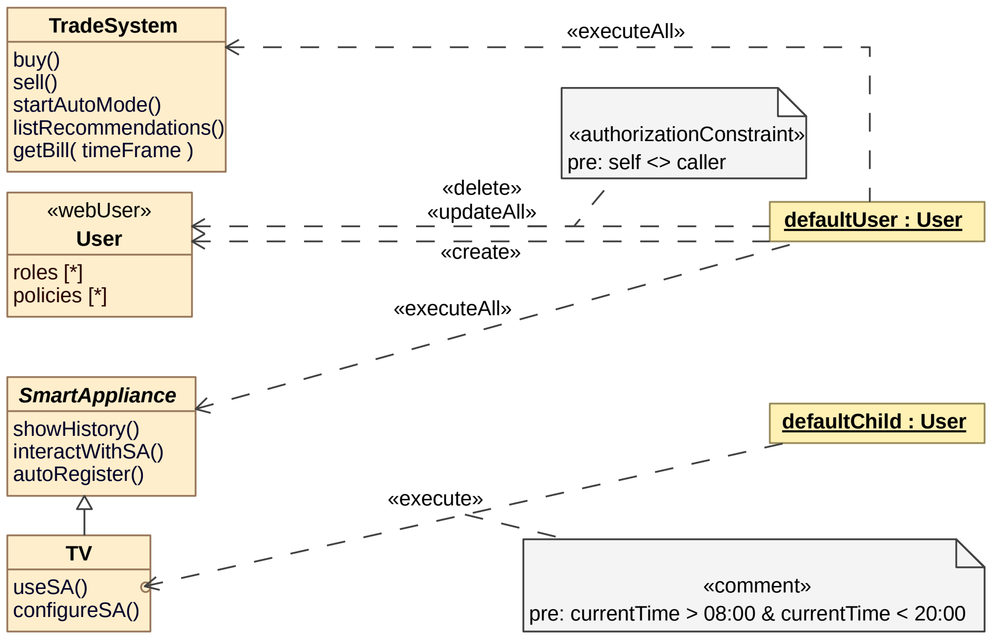


Figure 5.4: Example: Basic Rights (SmartGrid EMS)

Further information about the Basic Rights model and about common identifiers that are used in UWE models is provided in section 5.3.

5.2.4 Navigation Model

The Navigation model defines the navigation flow of the application as well as navigational access control policies. The former shows which possibilities of navigation exist in a certain state (usually from the view of a user[10]). The latter specifies roles that are allowed to navigate to a specific state and the action taken in case access is not granted, as described in section 5.3.2. In a web application such actions can be, e.g., to logout the user and to redirect to the login form or just to display an error message, as shown in figure 5.6.

Note that the state-based Navigation model replaces the former class-based Navigation model, therefore the model uses the stereotype ≪navigationStatesModel≫ instead of ≪navigationModel≫. An advantage of the newer Navigation (State) model is that, due to UML's state regions, it is clear which states are parallel, e.g., because two navigational nodes, i.e., parts of web pages that may change individually, are displayed at the same time. A disadvantage can be that UML state charts tend to provoke the feeling of a closed world, i.e. the feeling that all possibilities are defined. For web application's navigation, this turned out to be rather inconvenient due to the possibilities of the browser's back button, direct access via URL or changing application menus due to the subset of roles assigned to a user. For these purposes, the author defined shortcuts in her master's

[10]However, in practice GUI-related state machines can be enriched by important background-tasks that help to understand the underlying process logic. In this case, transitions can be stereotyped using ≪logicalLink≫ to express that a transition does not represent a navigation possibility. We recommend using UML activity diagrams or dedicated UML state charts for redefining application logic in further detail, to avoid overcrowding the navigation view.

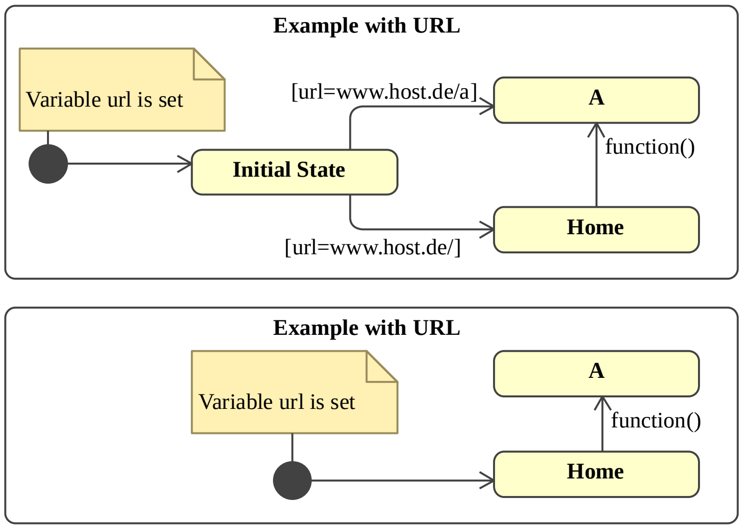


Figure 5.5: Navigation shortcut: a `url` variable holds the current URL, but the shortcut below abstracts from these details

thesis [37], as e.g., that a state in a «navigationStatesModel» that is not stereotyped can be seen as a «navigationalNode» (□). Figure 5.5 depicts a common shortcut for direct access via URL. The lower version abstracts from URLs.

In addition, some common actions can be highlighted by stereotypes, such as «search» (?) on UML transitions or «collection» (≡) on navigational nodes to express that items of the same kind are presented. For example, in figure 5.6, a collection of offers is presented to users. The tag {itemType} references the class `EnergyOffer` from the Content model. Outgoing transitions that are named with an underscore like `buyOffer(_)` are a shorthand for `buyOffer(item:ItemType)`. Further details can be found in [37, p.48].

5.2.5 Presentation Model

UWE's Presentation model sketches logical building blocks of a web application's user interface. Composite structure diagrams can be used to express composition as nested UML classes and properties. Figure 5.7 shows an excerpt of the SmartGrid Offers Application. On an offer creation web page, a form with a text field for a value can be submitted by pressing the `SubmitOffer` button. A confirmation page offers two links for accepting or canceling the submission of the new offer.

We prefer using mock-ups, as they are easier to create, closer to the look and feel of the resulting application and provide up-to-date widgets. During the work on this thesis, UWE Presentation diagrams were only used for the transformation from UWE to ActionGUI.

5.2.6 Application States Model

We defined an Application States model to distinguish general modes an application can enter from states, such as those from the Navigation model. States can be related to the application logic or to organizational purposes, like "running", "under Denial of Service (DOS) attack" or

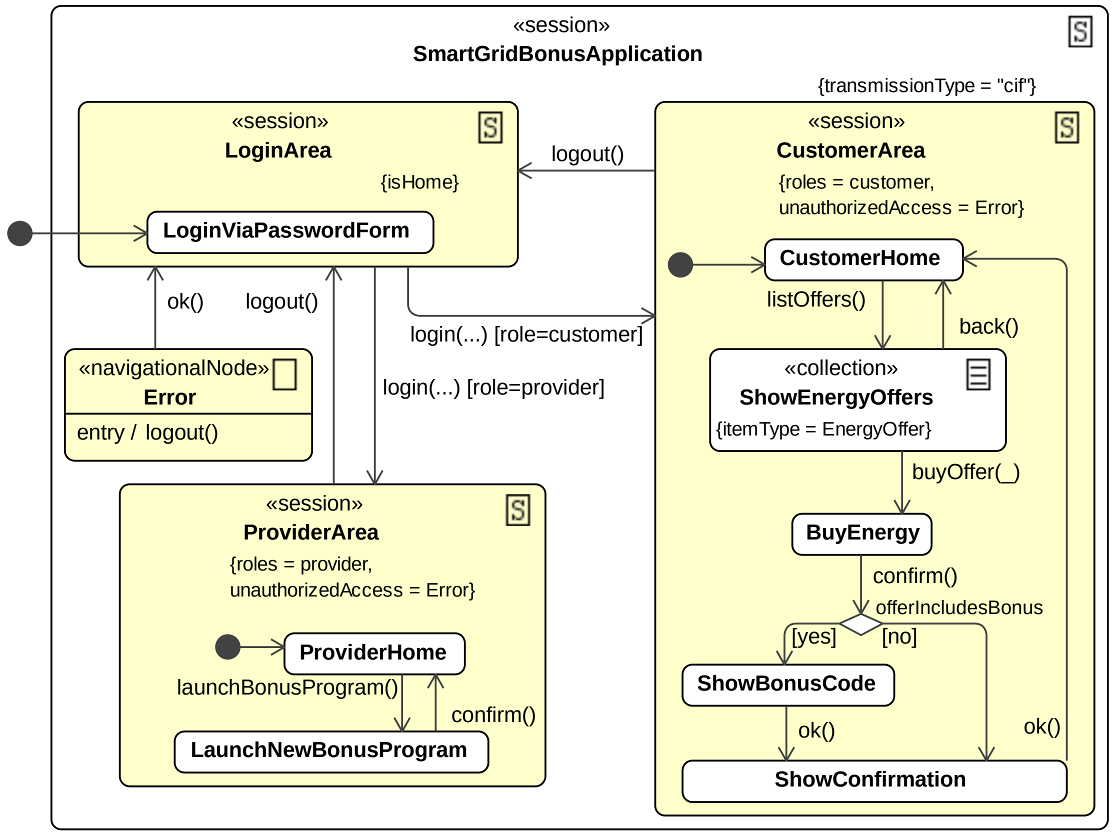


Figure 5.6: Example: Navigation (SmartGrid Bonus Application)

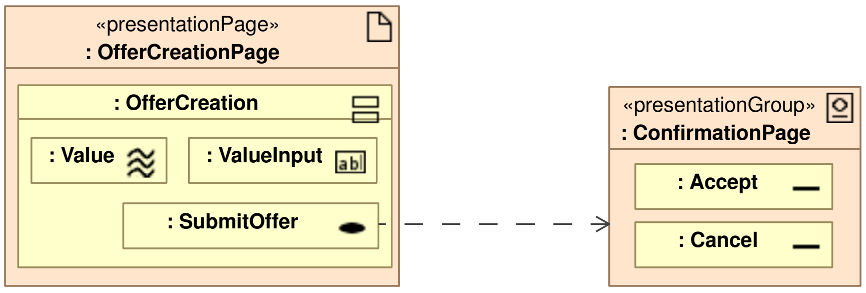


Figure 5.7: Example: Presentation (SmartGrid Offers Application)

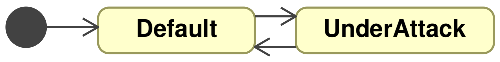


Figure 5.8: Example: Application States (SmartGrid EMS)

various maintenance modes. E.g., in our EMS case study, we used a default mode and an `UnderAttack` mode, as depicted in figure 5.8.

5.3 Security Extensions for UWE

This section describes UWE's model elements for modeling the requirements and the design of web applications and introduces common identifiers that can be used in string expressions. Note that the UWE profile contains much more model elements than introduced in this chapter, as we focus on security-related elements. Further information can be found, e.g., in [115, 116] and on the UWE website [224].

In the *requirements phase* (or phases) in the development process, the main task of software engineers is to analyze the whole scenario, as depicted in section 5.3. They elicit user's needs and have to find out which assets are influenced or managed by the software. Risk rating helps to discover security properties that should apply while the software is working with these assets. Unfortunately, some vulnerabilities result from errors so commonly made that these vulnerabilities tend to be in many applications that are not thoroughly overseen regarding security during development [161].

Modeling threats and thinking about the motive and ability of potential attackers can be simplified for applications that are online, as attacking common vulnerabilities can be automated to a large degree. Many attackers are not even interested in the kind of application they attack or the kind of data they could steal, but automatically scan for common vulnerabilities 24 hours a day to find, e.g., servers that can be taken over and used for criminal purposes like spamming, further vulnerability scanning, or (distributed) DOS attacks [92]. With UWE's security modeling elements, we focus on modeling assets and their security properties (cf. figure 4.2) together with

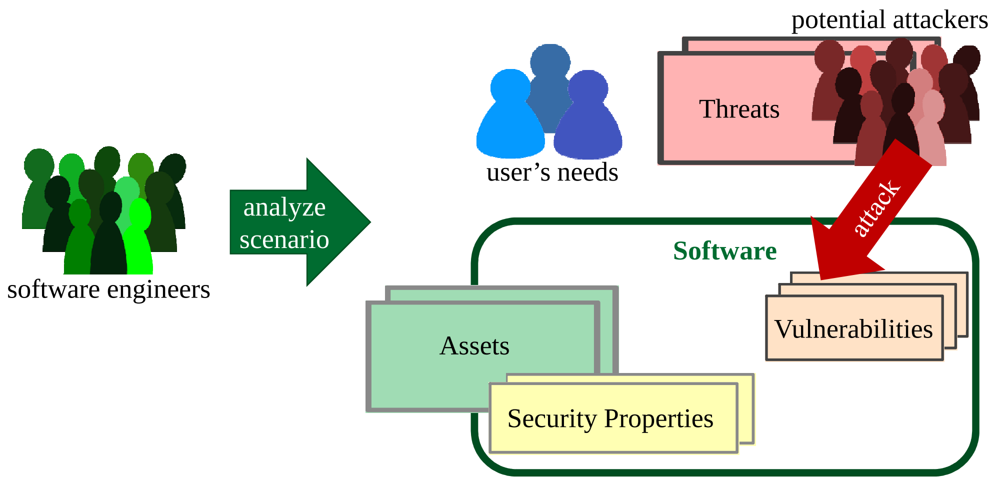


Figure 5.9: Secure software engineering: requirements

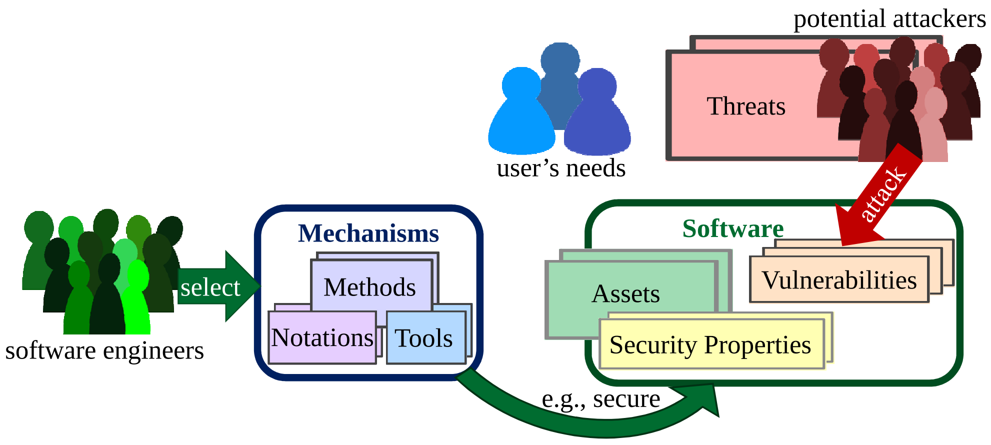


Figure 5.10: Secure software engineering: design

the web application's architecture. Concrete assets can be modeled using plain UWE (e.g., using components or classes in the Content model or states in the Navigation model). Assets can have security properties (cf. figure 3.3), which we express by using UML stereotypes and tags on the assets' UML elements. Therefore, the first subsection of this section is structured by the main security properties that we presented in SecWAO (cf. chapter 4).

In the *design phase* (or phases) of the development process, software engineers select mechanisms, i.e., methods, notations and tools. Section 5.3 depicts that mechanisms can "secure" the software under development, which means that required security properties hold and that the attack surface is minimized, which leads to fewer exposed vulnerabilities. Note that mechanisms can be selected and used in all activities of the development process. For example, it is common that tools that are used for testing are selected during testing activities and not during the design phase. However, in the design phase many mechanisms are selected that influence not only the implementation, but also define the context for running applications due to decisions regarding their configuration and needed environment. As all mechanisms are related to methods, the second subsection is structured according to SecWAO's main methods (cf. figure 4.4). The process of selection is usually preceded by a kind of evaluation process, which can be structured and documented using SecEval (cf. part II).

Mechanisms define how a security property, as e.g., confidentiality is realized. For example, it is not sufficient to demand that data should be kept confidential, but developers have to know which mechanisms they should employ. Modelers decide, how detailed the documentation with UWE is, so that supporting confidentiality could mean to model authorization (a `Method`), but it could also mean to define that a designated data should only be stored encrypted with a certain library (a `Tool`). One might think that software security is not important in the late design phase, as some vulnerabilities can be avoided by good programming practice and modern web application frameworks. However, "some of the most damaging web application vulnerabilities "in the wild" are still as widespread and just as damaging over 10 years after being discovered" [171, p. 1] and we experienced that our UWE extensions can bring application security more into focus.

Generally, security features are cross-cutting concerns that cannot be separated completely from a software's design. If detailed processes are considered, aspect-oriented modeling can be used for weaving restricting state machines into state machines that describe application workflows, as described by Zhang et al. [230]. Other attempts to model security using aspects are surveyed by Dehlinger and Subramanian in [67]. Note that for UWE "separation of concerns" does not refer to considering security separately, but to the different views on the system's architecture (cf. section 2.3.2).

We selected the following modeling elements according to SecWAO's main methods and security properties (cf. figure 1.3, lower arrow). To find beneficial modeling solutions, we experimented with our case studies to find out, which way of modeling is useful in practice. Note that although we demonstrate how to model requirement and design decisions using UML, most concepts could similarly be added to other modeling languages, as long as they are as expressive as UML.

For describing UWE's model elements, we use the following template:

«stereotype» **:** List of UML Metaclasses it can be applied to (UWE model it is used in)

{a tag that belongs to this stereotype **:** its type}

{another tag **:** its type}

In addition, these modeling elements are depicted in relevant excerpts of the UWE profile in appendix C, grouped by UWE model. For the sake of brevity, we favor small examples in this section instead of examples from our case studies. These and other examples that explain UWE's model elements can also be found in the UWE profile at the UWE website.

5.3.1 Modeling Required Security Properties

In chapter 4, which introduces SecWAO, figure 4.2 depicts the following security properties: system and data availability, system and control flow integrity, data confidentiality and integrity, data authenticity and freshness, user identity, user privacy (a vague term that is often related to anonymity and pseudonymity), non-repudiation, non-interference and data retention. As our goal with UWE is to be able to express common security properties and system's requirements at a rather abstract level, we do not include non-interference. For modeling non-interference in detail with UML state charts, the interested reader is referred to work from Ochoa et al. [154]. System integrity is also left out, because aiming at it is self-evident – even for systems that run honey pots, otherwise their logging functionality might be compromised. In many cases, requirements tend to be much more abstract than design decisions, which means that models are not only getting enhanced by design-related elements in section 5.3.2, but are also redefined over time, so that modeled requirements get replaced or deleted.

System Availability and Data Availability

Both, data availability and system availability are foundations for secure web applications. The application itself might be programmed accurately, but in case it is not available when needed, or in case the underlying data is not available, the application is useless.

«webUseCase» (⚙) : UseCase, Package (Content model)

{availability:String} importance of availability, availability goals, or planned measurements.

«storage» : Comment (Content model)

{availability:String} maps general data availability requirements to components, packages, classes or attributes. To extend UML comments reduces duplications, as one comment can be connected to several UML elements, as shown in figure 5.14.

≪component≫ : Component (Content model)

{scalability:String} planned measures to ensure scalability, e.g., through horizontal / vertical scaling.

In later phases of the SDLC, scaling can also be described in detail for a concrete scenario using UML nodes and node instances and describing their hardware requirements using comments.

Control Flow Integrity

Control flow integrity can be as difficult to grasp as system integrity. However, modeling can shine a light on valid control flows by making them explicit. For example, navigation-related control flows determine the paths a user can follow while browsing through a web application. Originally, UWE aimed at expressing navigation at a high level of abstraction, meaning that only the most important navigational nodes and navigation possibilities in between are modeled. However, the Navigation model also allows modeling all navigational paths of a critical part of an application in detail, such that they can be enforced at runtime. This is explained in further detail in section 6.5.

To know which part of the navigation is modeled exhaustively, the following tag can be set to a surrounding navigation state, i.e., a UML state machine or state:

≪navigationalNode≫ (▯) : StateMachine, State (Navigation model)

{enforceNavigationFlow:Boolean} is true if the navigation flow inside of the tagged element is modeled exhaustively and no other navigation possibilities should be allowed at runtime. An example is depicted in figure 5.11.

Data Confidentiality and Data Integrity

At the level of first requirements, it might not be clear, which data structure will be used for storing data. To express the requirements of data confidentiality or data integrity in general, a

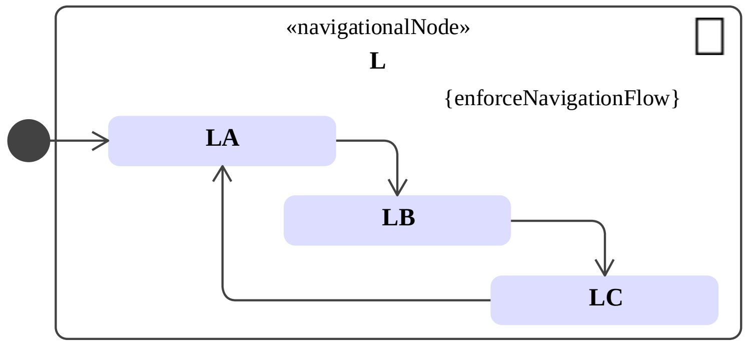


Figure 5.11: Enforcing navigation flow: The user cannot leave the predefined navigation path, i.e., the user cannot navigate to `LC` without navigating to `LB` directly before that.

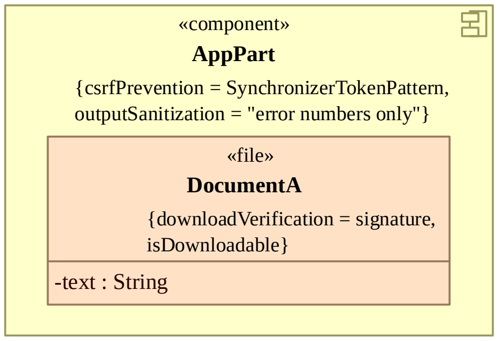

«component»
AppPart
{csrfPrevention = SynchronizerTokenPattern,
outputSanitization = "error numbers only"}
«file»
DocumentA
{downloadVerification = signature,
isDownloadable}
-text : String

Figure 5.12: Security in the Content model

first step can be to model a high-level activity diagram with UML object nodes that represent data in the UWE Requirements model. Another possibility is to model data as classes or attributes in the Content model. For both cases, data confidentiality or integrity can be modeled by connecting a comment:

≪storage≫ : Comment (Content model, Requirements model)

{confidentiality:Boolean} is true if the data should be stored and transmitted confidentially.

{integrity:Boolean} is true if the data should be stored and transmitted such that loss of integrity can be detected.

Modeling techniques for a more detailed view on confidentiality can be found in section 5.3.2, which covers several methods of authorization. For transmitting files[11] and checking their integrity after a user downloaded them, the following tags can be used:

≪file≫ : Class (Content model)

{downloadVerification:Verification} where predefined entries for the `Verification` enumeration are: hash, transportSecurity or signature (as used in figure 5.12). All these methods can help to ensure that a file a user downloaded kept its integrity.

{secureTransportOnly:Boolean} True means that confidentiality as well as integrity should be guaranteed by a protocol that provides transport security, as e.g., TLS.

To express that confidentiality and integrity are required for all data that is transmitted in a certain situation, the Navigation model can be used, as described in the following section.

Data Authenticity and Data Freshness

When implementing an application, the transmission protocol between servers and clients does not have to be modeled in detail, as it is just a building block to be used. Nonetheless, it is necessary to think about required security properties in order to be able to choose an appropriate protocol.

[11] A file can be tagged as downloadable ≪file≫ with the tag {isDownloadable:Boolean} or as stream with {isStream:Boolean}.

≪webUseCase≫ (⚙) : UseCase, Package (Requirements model)

> {transmissionType:String} similar, but usually described at a higher level of abstraction than the following tag:

≪navigationalNode≫ : StateMachine, State (Navigation model)

> {transmissionType:String} describes how data is secured that is transmitted while being in this navigational node. In the beginning, we used the abbreviation "cif" for confidentiality, integrity and freshness, but it turned out, that unabridged terms like "confidentiality, integrity, data authenticity and freshness" are more useful, as they are unambiguous.

After engineers reach a design decision upon which transmission method should be used, general requirements can be replaced by an encryption method like "TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256, 128 bit keys, TLS 1.2".

User Identity

A unique user in UWE is represented as a UML class with the stereotype ≪webUser≫. In UWE, we focus on users that can play various roles, thus a user can have not only attributes, but also roles (a default `Role` class can be used or copied from the UWE profile) and the user and the role are often also related to classes from the Content model. A ≪webUser≫ can be referred to as `caller` when defining access control (cf. section 5.3.2).

≪webUser≫ : Class (Role model)

> {verifiedIdentityBy:String} Methods to verify the user's identity in the first place (related to registration, cf. paragraph "Authentication and Session Management" in section 5.3.2). Common methods are depicted in the lower left corner of figure 4.6 in section 4.3.

User Privacy: Pseudonymity and Anonymity

As user privacy is too wide a subject to be modeled in all its facets in UWE, we concentrate on pseudonymity and anonymity. When creating a web account, several properties can be distinguished:

≪webUser≫ : Class (Role model)

> {pseudonymity:Boolean} True means that users do not have to use their real names. Relates to {verifiedIdentityBy}, cf. previous section.
>
> {enforceAnonymity:String} describes requirements users must fulfill regarding the network they use. An example would be a web service that should only be accessible by TOR[12] users.

It is worth mentioning that classes that are stereotyped by ≪webUser≫ can have attributes for a name or a pseudonym or other user-related data. However, omitting user attributes in a web application does not mean that no user-related data is collected. E.g., IP addresses can often be found in web servers' logfiles. It is worthwhile noting that it has to be decided towards whom anonymity or pseudonymity should hold, which goes hand in hand with user-defined access control (cf. section 5.3.2).

[12]TOR. https://www.torproject.org/

Non-Repudiation

A basic requirement for implementing non-repudiation is having a trusted third party. It can be modeled as a UML node in a deployment diagram using the UWE stereotype ≪trustedThird-Party≫. To use the feature of non-repudiation for an operation execution or for attribute access, the following tag can be used on dependencies that connect roles to elements like classes, attributes or operations from the Content model, as shown in figure 5.13:

≪accessType≫ : Dependency (Basic Rights model)

{nonRepudiation:trustedThirdParty} A ≪trustedThirdParty≫ node can be referenced. A request should only be processed any further, if the trusted third party has logged it.

Note that in this context non-repudiation does not apply to messages, but to the concept of access. For its implementation, logging failures and successes by the trusted third party after the process can be advisable to be sure that access had been granted. For operations in UWE we model this directly on operations with the following stereotype:

≪non-repudiation≫ : Operation (Content model)

{trustedThirdParty:trustedThirdParty} A ≪trustedThirdParty≫ node can be referenced. As documenting successful execution is a complex task, we assume that operations are used, which means that e.g., for logging successful access to attributes getters or setters have to be introduced.

Data Retention

Data retention defines which information is stored and how long it will be kept.

≪storage≫ : Comment (Content model, Requirements model)

{retention:String} defines the time span data should be kept. It is advisable to point to an external documentation that includes all cases, as e.g., data retention in the case that users request to delete their accounts.

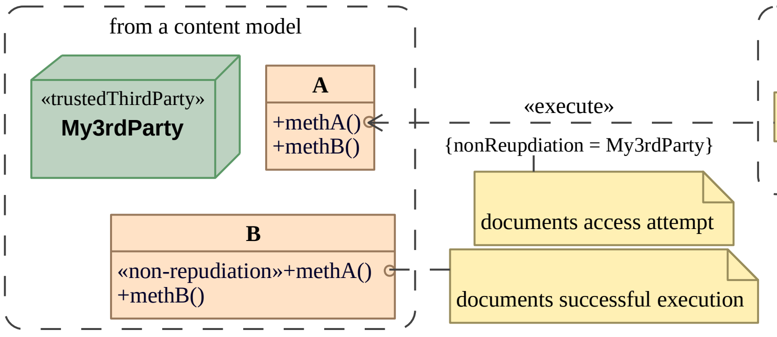


Figure 5.13: Non-Repudiation: Executing `methA` is allowed for `roleX` and the attempt is documented by the trusted third party `My3rdParty`. (The stereotype ≪execute≫ is a descendant of ≪accessType≫, cf. "Authorization" in section 5.3.2).

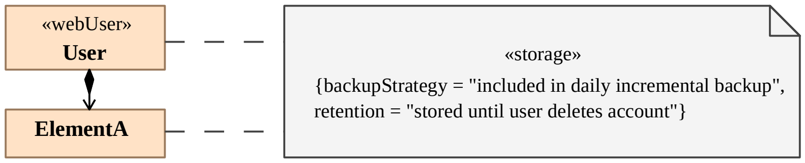

«webUser»
User
ElementA
«storage»
{backupStrategy = "included in daily incremental backup",
retention = "stored until user deletes account"}

Figure 5.14: Data retention: the comment applies to user data and data of `ElementA`

{backupStrategy:String} Having a backup strategy is not only important in case of data loss, but also interacts with the retention strategy, as in some cases, laws like the Bundesdatenschutzgesetz[13] can make it necessary to delete or block the access to all occurrences of personal data.

5.3.2 Modeling Applied Mechanisms

Figure 4.4 from chapter 4 depicts the following methods: authentication, session management, authorization, data validation, error handling, logging, system configuration and cryptography. In this section, we describe how to document selected methods together with the application's design using UWE.

Authentication and Session Management

In her master's thesis [37], the author modeled patterns for common authentication scenarios, such as registration for an account, login and credential recovery processes. We added a pattern for listing remote sessions that allows users to invalidate their sessions and several patterns to model how a user can be verified, as shown in figure 4.6 in section 4.3. Additional to remote session invalidation, another method preventing the takeover of active sessions is CSRF prevention (cf. figure 5.12):

≪component≫ : Component (Content model) and
≪session≫ ($\boxed{S}$) : StateMachine, State (Navigation model)

{csrfPrevention:CSRFprevention} where `CSRFprevention` is an enumeration that lists common strategies for CSRF prevention, as depicted in figure 4.6. We hope that tags like this quickly become obsolete, because modern web frameworks take care of CSRF per default.

A session can be modeled in detail using the stereotype ≪session≫ ($\boxed{S}$):

≪session≫ (descendant of ≪navigationalNode≫) : StateMachine, State (Navigation model)

{sessionData:sessionClass}, where `sessionClass` can be modeled in the Content model to show which session-related information is held.

{reauth:Boolean} True requires a user to re-authenticate when entering this state.

{reauthIf:String} re-authenticates a user when entering this state, but only, if the given expression does evaluate to true. For example, re-authentication should only be required, if the previous authentication process is longer ago than 15 minutes.

[13]Bundesdatenschutzgesetz. http://dejure.org/gesetze/BDSG

{newSessionID:Boolean} True means that a new session identifier is used. Note that session states that are nested in other session states do not start a new session by default, unless the state of authorization changes from unauthorized to authorized (which is necessary to avoid session fixation attacks).

The login and logout process can be indicated in the Navigation model by labeling transitions. We suggest using `login()`, `logout()` and `after(time span)/logout`, which is used as an equivalent for `idle(time span)/logout`.

Authorization

In the following, we cover general authorization issues, access control rules for content elements (i.e., classes, attributes and methods), and rules for navigational nodes (cf. figure 4.7).

General Authorization. The following two tags may serve as a reminder for complete mediation[14], i.e., the principle that access control rights have to be checked for each access.

≪component≫ : Component (Content model)

{objectRefManager:String} decides for a central component or library with the aim to avoid insecure direct object references [161], i.e., direct access to data while bypassing the authorization system.

{accessControlEnforcementSystem:String} generally states which access control system is used. Examples are tools like Spring [91], or DRM enforcement systems.

The tags {confidentiality} and {integrity} of the ≪store≫ stereotype introduced in the paragraph “Data Confidentiality and Data Integrity” in section 5.3.1 can be defined in further detail:

≪storage≫ : Comment (Content model, Requirements model)

{confidentialityStrategy:String} The strategy can be, e.g., access via an access management system or the deployment of encryption. It usually elaborates on the general information given in {accessControlEnforcementSystem} of ≪component≫.

{encryption:Boolean} is true if targeted data may only be stored and transmitted in an encrypted way.

{integrityStrategy:String} describes the strategy for integrity more precisely.

For files, an arbitrary number of Digital Rights Management (DRM) actions can be allowed:

≪file≫ : Class (Content model)

{drm:DRM} where literals of the DRM enumeration include the following actions for digital rights management: view, delete, copy, share, print, save and extractContents.

Access Control Rules for Content Elements. The main stereotype for defining access control rules for elements from the Content model is the ≪accessType≫ stereotype, which serves as an ancestor for allowed actions. The stereotyped dependency is designed to connect a source (usually a role instance) with a target (usually a class, attribute or method).

[14]Complete Mediation.
https://buildsecurityin.us-cert.gov/articles/knowledge/principles/complete-mediation

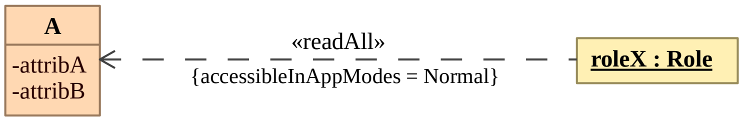


Figure 5.15: Restricting access to application modes: here to a mode called “Normal”

≪accessType≫ : Dependency (Basic Rights model)

{accessibleInAppModes:appMode} References application modes, in which access to the target is allowed, as depicted in figure 5.15.

{notAccessibleInAppModes:appMode} References application modes, in which access to the target is *not*[15] allowed.

Common descendants of ≪actionType≫ describe actions on the following targets:

attributes: ≪read≫, ≪update≫

methods: ≪execute≫

classes: ≪delete≫, ≪create≫ and ≪readAll≫, ≪updateAll≫, ≪executeAll≫. Stereotypes ending with “All” refer to all attributes or methods of a class, except those that are specified using a tag called {except}[15].

Access can be defined in more detail using UML comments with the UWE stereotype ≪authorizationConstraint≫.

Usually, defined actions are “or” actions, which means that if two roles are allowed to, e.g., execute a particular method, they can do so independently. However, there are situations, where more than one user should be needed to perform an action. This concept is called separation of duties[16] or many-eyes principle. In UWE, we model that a role can demand to execute an action using the stereotype ≪separationOfDuties≫:

≪separationOfDuties≫ (descendant of ≪authorizationConstraint≫) : Comment (Basic Rights model)

{timeFrame:String} The final execution of an action is deferred until eligible users have requested it within the given time frame. The time frame starts with the first request and ends with the successful access or the elapse of the given time.

{roles:String} The stereotype ≪separationOfDuties≫ can define eligible users in two ways, as depicted in figure 5.16:

- If the {roles} tag is not set, the stereotyped UML comment can be connected to two or more dependencies with ≪actionType≫ stereotypes. This means that all connected actions have to be requested at least once for the action to be executed. Note that the requested ≪actionType≫ stereotype has to be in the set of stereotypes of each connected dependency.

[15] Use with care, as it contradicts a positive security model, which defines allowed actions instead of defining exclusions.
[16] Wikipedia: Separation of Duties. https://en.wikipedia.org/wiki/Separation_of_duties

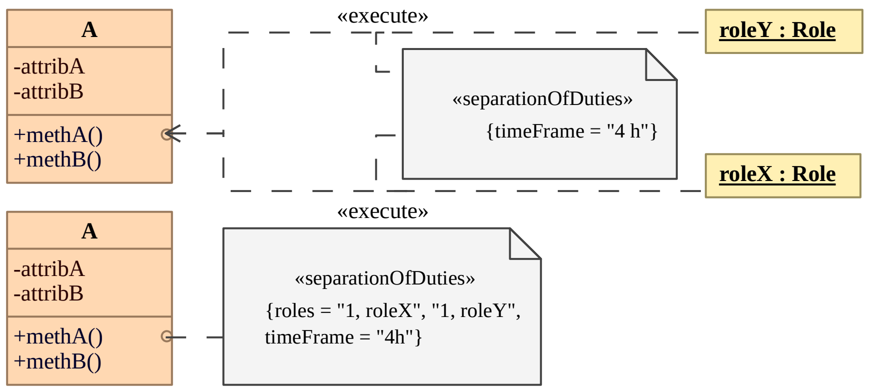


Figure 5.16: Separation of duties: In both modeling alternatives, the function `methA()` is executed, if two users playing roleX and a user playing roleY both demand its execution within four hours

- If the {roles} tag is set, the stereotyped UML comment has to be connected directly to the target that should be accessed. A flexible way of adding eligible users is to use a list with entries formatted like “number of unique users, role name”, which means that a number of unique users from a certain role has to request the action. If no number is given, it is assumed to be 1.

For the sake of consistency with SecWAO (figure 4.7), we also created a stereotype with tags for the following data processing actions:

≪dataProcessing≫ (descendant of ≪accessType≫) : Dependency (Basic Rights model)

{aggregating}, {analyzing}, {classifying}, {summarizing}, {reporting}, {validating} and {sorting} Although we do not use them in our case studies, they might be useful as shortcuts for distinct methods within a software project with a focus on data processing.

Many web applications allow their users to co-determine access rights for other users on parts of their data. To clarify on which data the access can be user-defined and to set granularity and to specify default actions, the following tags can be used. Note that the source of the dependency can be arbitrary, in case it represents a user-defined group of users or roles.

≪userDefined≫ (descendant of ≪accessType≫) : Dependency (Basic Rights model)

{selectionRange:Enumeration} references an enumeration whose literals represent actions that can be selected by users to grant access to the target, as shown in figure 5.17.

{defaultSelection:EnumerationLiteral} specifies a default selection in case the user does not personalize anything. The literal has to be part of the enumeration that is referenced by {selectionRange}.

Access Control Rules for Navigational Nodes. Theoretically, it would be possible to model access to data during data transmissions or access to navigational nodes in the Basic Rights model. In practice, it turned out to be useful to model both mainly in the Navigation model, although this does not mean that data has to be transmitted over the internet for each state change that occurs in the navigation.

navigationalNode : StateMachine, State (Navigation model)

> {accessibleInAppModes:appMode} references application modes, in which access to this state should be allowed.
>
> {notAccessibleInAppModes:appMode} References application modes, in which access to this state should not be allowed.
>
> {accessPrecondition:String} Access is only granted, if the given expression evaluates to true. In previous versions this tag was called "rolesExpression", as precondition often depended on roles a user plays.

As the most common case for access control is to grant access according to a certain role a user plays, the following tag was introduced in [37] as a shortcut for (user.roles→includes(r_1) | ... | user.roles→includes(r_n)), where r_1, ... , r_n are a list of role names for which access should be allowed.

≪session≫ (descendant of ≪navigationalNode≫) : StateMachine, State (Navigation model)

> {roles:InstanceSpecification} To access a node, the users must have at least one role in their assigned roles that also exists in the set of the {roles} tag. An example is shown in figure 5.6.

In the paragraph "Error Handling", we introduce tags for pointing to an error state that gets activated in case access is denied on a navigational node.

Data Validation

Many common vulnerabilities in web applications are related to insufficient data validation, as e.g., SQL injection, Cross-Site Scripting (XSS) or information disclosure via error messages or forged logfile entries.

≪component≫ : Component (Content model)

> {inputValidation:String} states how untrusted input should be treated in this component.
>
> {inputValidationType:InputValidationType} where `InputValidationType` contains the enumeration literals "blacklisting" and "whitelisting" to describe the general method that should be used, e.g., especially to raise the awareness for locations where blacklisting is used and thus has to be maintained.
>
> {outputSanitization:String} describes how output should be sanitized, which may include guidelines for logfile entries and error messages. The simple example in figure 5.12 states

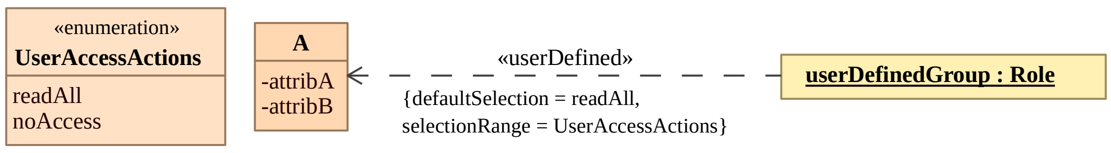


Figure 5.17: User-defined access control: The default right for a user-defined group is to read all attributes of class `A`. Users can change this and grant "noAccess" instead.

that only error numbers should be shown, which can have a severe impact of usability, but may be useful for critical core components.

{sqlInjPrevention:SQLinjectionPrevention} where `SQLinjectionPrevention` is one of the common three countermeasures to SQL injection, as already depicted in SecWAO, at the bottom of figure 4.5 in section 4.3.

{injectionPrevention:String} documents other countermeasures for injection attacks.

{xssPrevention:XSSprevention} where `XSSprevention` is a selection of methods we already introduced in section 4.1.

Unfortunately, UML does not allow to specify that both enumeration values and as strings are allowed as value types. Thus, we decided that it is more important for us to be able to select predefined items from an enumeration than to enter free text. Free text (e.g., for specifying a library that should be employed) can be added additionally by using a UML comment.

Error Handling

Handling errors is a task that is immanent to software development. Errors can range from hardware failures over software failures, successful attacks, up to error cases in the application logic. The Application States model can be helpful to structures general modes an application can be brought into and to define actions that should be taken in certain states and conditions that lead to transitions. Regarding user navigation, typical error cases result from access requests while having insufficient permissions:

navigationalNode : StateMachine, State (Navigation model)

{unauthorizedAccess:navigationalNode} the navigation state that should be navigated to in case access to the current state is prohibited.

In case the target state is shown nearby in the same diagram, a transition stereotyped `unauthorizedAccess` can be used instead, as shown in figure 5.18.

Logging

Writing logfile entries is often associated with application logic, application state changes or reporting errors. For general purposes, we introduce the stereotype ≪logged≫ on UML comments. For logging actions, the tag {logged} can be set to true.

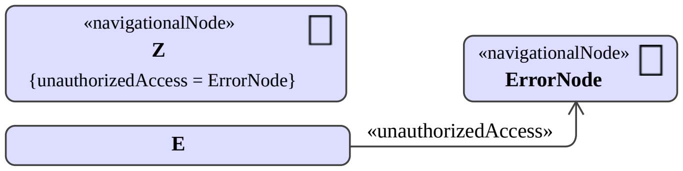


Figure 5.18: Unauthorized access: In case unauthorized access to navigational nodes `E` or `Z` occurs, the application should navigate to `ErrorNode`.

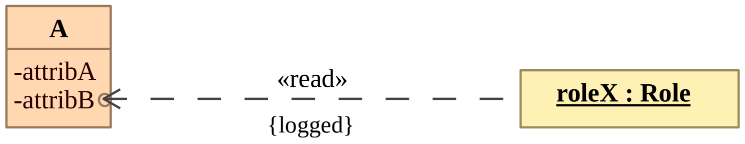


Figure 5.19: Logged access: access attempts are logged

≪accessType≫ : Dependency (Basic Rights model)

{logged:Boolean} is true if a log entry should be written for each access attempt, no matter if successful or not (cf. figure 5.19). Note that this does not guarantee non-repudiation, as for non-repudiation, an infrastructure with a trusted third party is needed, cf. paragraph Non-Repudiation in section 5.3.1.

System Configuration and Cryptography

It is not always easy to distinguish decisions in the requirements and design phases that will affect the code itself from those that will have effects on the application or system configuration. Common locations for hints regarding the system and web server configuration can be found in availability or scalability tag descriptions (cf. section 5.3.1) and in the {transmissionType} description of navigational nodes. Additionally, a strategy for software updates can be set:

≪component≫ : Component (Content model)

{updatePolicy:String} refers to a policy describing how the component is going to be updated.

UWE does not aim at modeling cryptographic algorithms. However, UWE can model cryptographic methods that should be deployed, e.g., for data transmission (cf. paragraph "Data Authenticity and Data Freshness" in section 5.3.1), or for encrypting files (cf. section 5.3.2).

5.3.3 Identifiers in UWE

Although expressions in UML can be written in OCL, OCL was poorly received in practice and it has its limitations regarding real-world scenarios, as it mostly requires a closed-world assumption, which means that every detail needs to modeled. For web applications, modeling every detail in UWE in many cases is an absurd waste of time, especially as the main advantages of UWE models for web applications turned out to lie in their clarity and comprehensibility that can be achieved by abstraction. Thus, in order to be useful, expressions have to be quickly understandable by humans, although best practices regarding common identifiers can help both, the humans and algorithms that parse UML diagrams. While modeling with UWE, the following identifiers turned out to be useful:

caller or user. Current instance of a content class that is stereotyped by ≪webUser≫, which is navigating through the web application.

role. Shortcut for, e.g., user.roles→includes(r) – or similar, according to the structure of user or role classes, which can then be used as role=r in guards or other expressions.

self or this (in the Basic Rights model). Refers to the target element, i.e., to the element the expression constrains access to.

url. A variable that contains the current URL the browser navigated to.

request.region. Geographical location of the region a request comes from (although it can easily be changed by proxies).

time or currentTime. Current server time.

date or currentDate. Current server date.

pwd.type. Can be "normal" or "panic". Panic refers to panic mode, where a user is threatened to use the application and enters a predefined password for a panic mode. Note that "pwd" can be seen as an abbreviation for a User class containing a UML role name called "pwd" that refers to a class that manages password metadata.

appModes.currModes. The current modes the application is in. Note that this does refer to general application modes from the Application States model, not to modes like the panic mode, which is individually be entered by a user.

appModes.StateName. `StateName` refers to an existing state, called "StateName" in the Application States model.

element.timesOfAccessWithin(timeSpan). Number of times the element was accessed during the given time span.

5.4 Summary and Related Work

In this chapter, we presented our modeling approach for secure web applications using several case studies: SmartGrid EMS, SmartGrid Offers, SmartGrid Bonus, a patient monitoring application, a hospital information system and ownCloud, which is an open-source web application. Major UWE models are the Requirements model, Content model, the Role model and the Basic Rights model, Navigation model, Presentation model, and the Application States model. The expansion of the UWE profile with security-related model elements and patterns gives an answer to the first part of RQ3: "How can security aspects of web applications be expressed?". UWE's security features filled the gap between general security requirements and mechanisms and concrete requirements and mechanisms relevant for a web application in practice. Thereby, our ontology SecWAO served us well for structuring UWE's stereotypes and tags. The full case studies and the UWE profile can be downloaded from the UWE website [224].

UWE's security extensions. The UWE profile enables web designers and engineers to model security properties and mechanisms together with functional issues. This serves as a means of decision making and documentation.

To the best of our knowledge, we are the first who provide means to systematically model used security methods and required security properties for web applications. However, modeling applications and access control rules is not new [199, 33], as we have already discussed in section 2.3.2. For example, the SecureUML approach [125] is similar to the Basic Rights model, although the latter specifies role-based access control using dependencies instead of association classes. This avoids the use of method names with an access-related return type. However, UWE's Basic Rights models can easily be transformed into a SecureUML representation, as long as no UWE-specific identifiers are used in authorization constraints. ActionGUI [11] uses SecureUML and although complete web applications can be described in detail with ActionGUI, most security methods cannot be expressed in models, but are predefined by the code generator or its target framework, which severely limits the variety of resulting web applications.

Outlook. It turned out that most UWE modeling concepts can also be used for other types of software than for web applications alone. For example, mobile applications and desktop applications can also transmit data over connections that have to be secured, authentication via online services is common and many applications incorporate web-like navigation structures. In the end, the differences of web, mobile and desktop applications become increasingly blurred; a trend which is accelerated by frameworks that allow for developing an application for several architectures, mostly by making use of HTML5. Examples for these frameworks are jQuery Mobile[17] or Apache Cordova[18].

[17] jQuery Mobile. https://jquerymobile.com/
[18] Apache Cordova. https://cordova.apache.org/

Chapter 6

Artifact Generation

In order to pursue our second objective of RQ3 – which is to examine how resulting security models can be used in the development process – we present various possibilities to generate artifacts that can be used in implementation and testing activities. The main idea is to generate helpful security artifacts from models representing a web application at a rather high level of abstraction, so that the models can remain small enough for practitioners to be able to diminish inconsistencies better than in code or other artifacts like configuration files.

Already in a master's thesis supervised by the author in 2012, Wolf [228] tested how UWE's Basic Rights model and Navigation model can be transformed so that the (navigational) access control rules that are specified can directly be added to an implementation. In that work, we used frameworks supporting separation of concerns, namely Apache Wicket[1] (a Java web application framework), Apache Shiro[2] (for implementing access control), Hibernate[3] (for persisting application data) and Google Guice[4] (for dependency injection). As this test turned out to be successful, we continued to aim at the generation of artifacts, instead of all-in-one solutions for generating code, which are rather inflexible and unsuitable for web applications due to the fast advancement of technologies.

In this chapter, we sketch how UWE can be represented textually and how UWE models can be transformed into this textual representation. Additionally, we introduce two toolchains related to access control and have a look at a transformation to the modeling method ActionGUI. Finally, we demonstrate the enforcement and testing of secure navigation paths. All generation algorithms presented in this chapter take at least the UML containment tree[5] of UWE models as an input.

[1]Apache Wicket. http://wicket.apache.org/
[2]Apache Shiro. http://shiro.apache.org/
[3]Hibernate. http://hibernate.org/
[4]Google Guice. https://github.com/google/guice
[5]The containment tree comprises UML modeling elements and their properties, but provides no information about the graphical layout of UML diagrams.

6.1 TextualUWE: A Domain-Specific Language

TextualUWE is a textual alternative to the graphical UWE notation, which is easy to read for humans and to exploit by algorithms. It was created in 2013, with the thought in mind that UML with its inaccuracies might not be the best choice for modeling security. However, it turned out that at the high level of abstraction that UWE models are located at, possible model inconsistencies due to the nature of UML do not play a major role. Nonetheless, we provide some background, briefly sketch TextualUWE's structure, validation possibilities and a transformation from UWE to TextualUWE. This section extends the description we provided in [205] and uses the SmartGrid Bonus application (cf. section 5.1.1) as an example.

6.1.1 Background

On the one hand, machine-readable modeling languages like XML Metadata Interchange (XMI) exist. Most times, XMI is used to export models (i.e. the tree of modeling elements and their properties) in order to analyze them or use them as input for another tool. Unfortunately files in the XMI format tend to be very complex and long, and thus hard to read.

On the other hand, Domain-Specific Languages (DSLs) and web frameworks are used that aim at producing executable software, usually providing only a rather low level of abstraction from technical details. Examples are WebDSL [93], a DSL from which Java web applications can be generated; Jif (Java + information flow) "a security-typed programming language that extends Java with support for information flow control and access control"[6] and a similar framework for web applications, called Sif [55]); and web security frameworks that claim to be resistant to common vulnerabilities, as e.g., Lift[7] for web applications written in Scala; Shiro[8] for Java; or HDIV[9] for Spring, Grails, Struts and JSF. The OWASP project "OWASP Secure Web Application Framework Manifesto"[10] started to list security requirements for developers of web application frameworks. Despite efforts to adhere to the principle "secure by default" (cf. Microsoft SDL [124]), most security methods and tools have to be configured and to be used on purpose, as security design decisions have to be taken prior to using them, as e.g., not every web application needs authentication.

6.1.2 Structure

As we started by pursuing the idea of abstract models that are better readable than XMI, but less technology-dependant than existing web frameworks or DSLs, we decided in favor of a direct textual equivalent to graphical UWE models and created an internal DSL. "An internal DSL is a DSL represented within the syntax of a general-purpose language. It is a stylized use of that language for a domain-specific purpose" [85, ch. 1]. As we want to open the way to expressive algorithms, we decided to use Scala[11], a multi-paradigm programming language that supports

[6]Jif. http://www.cs.cornell.edu/jif/
[7]Lift. http://liftweb.net/
[8]Apache Shiro. https://shiro.apache.org/
[9]HDIV. http://www.hdiv.org/
[10]OWASP Secure Web Application Framework Manifesto. https://www.owasp.org/index.php/OWASP_Secure_Web_Application_Framework_Manifesto
[11]Scala. http://scala-lang.org/

object-oriented programming in both functional and imperative style. We restricted our approach to Scala's functional style, because of its conciseness. The data structure for our DSL is also plain Scala so that there is no need for a special DSL editor.

The textual version of UWE is located in two Scala packages: one specifying the DSL and another one containing functional verification algorithms. Classes, attributes and methods in UML are translated into Scala classes, values and function definitions. For additional information, e.g., from UWE stereotypes and tags, we define Scala annotations[12]. As an example, we detail how the Navigation model is described by TextualUWE.

As the Navigation model is based on a UML state machine, we have to express state machines, states and transitions. In addition, stereotypes can be set on states and transitions. States can be "simple" or "composite". Composite states contain state machines which are executed in parallel. The following listing shows an excerpt of our definition:

```
object NavigationStateMachine {
  sealed abstract trait State
  case class SimpleState(name: String,
                 stereotypes: Set[StateStereotype] = Set()) extends State
  case class CompositeState(name: String,
                    regions: Set[StateMachine],
                    stereotypes: Set[StateStereotype] = Set()) extends State

  case class Transition(source: State,
                 target: State,
                 leftCStates : Int = 0,
                 guard: String = "",
                 stereotypes: Set[TransStereotype] = Set(),
                 enteredCStates : List[CompositeState] = List())

  case class StateMachine(initialState: State, transitions: Set[Transition])
  ...
}
```

Transitions connect two states and additionally they record which composite states were entered and how many composites states were left by this transition (which is denoted by the parameter `leftCStates`). This is necessary due to the nesting of state machines within composite states.

Already in section 5.2 we depicted the Navigation model of our SmartGrid Bonus application in figure 5.6. The SmartGrid Bonus application was briefly introduced in section 5.1.1: The application, represents a prototype of an energy offer management including optional bonus handling. It provides two different user roles namely *Provider* and *Customer*: Providers manage and sell energy packages including optional bonus programs for customers. Customers have the possibility to buy offered energy packages and to get associated bonus codes. Details of the customer's navigation possibilities are omitted at this point, because they do not contribute to the understanding of TextualUWE.

The textual version of our example is listed in the following.

[12]Scala Annotations. http://docs.scala-lang.org/tutorials/tour/annotations.html

```
object NavStateSmartGrid {
  val loginViaPF = SimpleState("LoginViaPasswordForm")
  val loginArea = CompositeState("LoginArea",
     Set(StateMachine(loginViaPF, Set())),
     Set(new NavigationalNode(true)))

  val customerArea = SimpleState("CustomerArea",
     Set(new Session(Set("customer")))) // shortened

  val providerHome = SimpleState("ProviderHome")
  val launchNewBonusProgram = SimpleState("LaunchNewBonusProgram")
  val innerTrans = Set(Transition(providerHome, launchNewBonusProgram),
             Transition(launchNewBonusProgram, providerHome))
  val providerArea = CompositeState("ProviderArea",
     Set(StateMachine(providerHome, innerTrans)),
     Set(new Session(Set("provider"))))
  val errorState = SimpleState("Error")

  val interTrans = Set(
   Transition(providerArea, errorState, 0, "unauthorized"),
   Transition(customerArea, errorState, 0, "unauthorized"),
   Transition(errorState, loginArea),
   Transition(providerArea, loginArea),
   Transition(customerArea, loginArea),
   Transition(loginArea, providerArea, 0, "role = provider"),
   Transition(loginArea, customerArea, 0, "role = customer"))

  val ts = interTrans ++ innerTrans
  val outerState = CompositeState("SmartGridBonusApplication",
     Set(StateMachine(loginArea, ts)),
     Set(new Session(Set(), "", TThsts)))
  val sm = StateMachine(outerState, ts)
}
```

6.1.3 Validation

Additionally, we worked on algorithms to check security features of TextualUWE models, as e.g., which part of the web application can be reached by a user which is associated to a certain role. Further verifiable features are to find inconsistencies in the model or to check what happens when navigational nodes are illegally accessed.

For the verification of model features, we use Scala functions. An example is the query which states can be reached if the user takes on certain roles (or other conditions are met, which have to be defined separately). The algorithm is simple: each state can be a simple state or a composite state. For simple states, an inner function makes use of sentry functions to decide whether or not the state is accessible. Sentry functions can, e.g., test for roles, or they can allow to enter all transitions and states, which results in a list of all reachable states, regardless of the roles, a user plays. For complex states, each state machine is examined.

6.1.4 Transformation from UWE to TextualUWE

A prototypical transformation for converting graphical UWE models into TextualUWE models was implemented in a bachelor thesis [181] under the supervision of the author. For the transformation, we used the Acceleo framework[13], which is an open-source text generator that implements a template-based approach for extracting information from input models. Thus, existing UWE diagrams can automatically be transformed into TextualUWE. In theory, the other way around is also possible and needs to employ diagram layout algorithms.

6.2 UWE2FACPL Toolchain: Generating Access Control Policies

The aim of modeling access control rules with the Basic Rights model is to have them enforced in a piece of software they were defined for. One way of enforcing access control is to use a Policy Decision Point (PDP), "which evaluates access requests against authorization policies before issuing access decisions"[14]. We present the transformation of access control rules from the Basic Rights model into two kinds of textual policies, namely XACML and FACPL. Thus, a PDP can use the generated policy for deciding at runtime which requests are permitted or not.

In this section, which is based on joint work with Nora Koch, Massimiliano Masi, Rosario Pugliese, and Francesco Tiezzi [43], we introduce a model-driven process that transforms access control policies from the UWE Basic Rights model to XACML. These XACML policies are then translated into FACPL, a policy language with formal semantics, and the resulting policies are evaluated by means of a Java-based software tool, as depicted in section 6.2.

In the following, we present background and the two transformations UWE2XACML and XACML2FACPL, before coming to the evaluation of policies and access requests. As an example, we use our Hospital Information System case study that has already been introduced briefly in section 5.1.3. Detailed policies and our tools can be found at the UWE web page [224].

6.2.1 Background

For this toolchain, two tools are used, which we briefly introduced in section 2.3.2: MagicUWE, which eases modeling UWE in MagicDraw [147] and the Service Development Environments (SDEs) [193], which is a tool workbench in eclipse [74] that allows executing integrated tools in a row. Many tools that were used in the NESSoS project are integrated in the SDE. Beyond others, we added the tools for web vulnerability scanning we compared in section 3.4, namely Arachni, Nikto, Nessus and Nexpose [122]. Figure 6.6 shows a screenshot of a toolchain in the SDE. Integrated tools can be downloaded from the NESSoS web page [193].

The XACML Standard

XACML permits decoupling the access control from the application's flow. In its underlying access control model, the access to each resource is regulated by one or more policies, i.e. XML documents express the capabilities and credentials that a requestor must have to access a resource.

[13]Acceleo. http://www.eclipse.org/acceleo/
[14]Wikipedia: XACML. https://en.wikipedia.org/wiki/Xacml

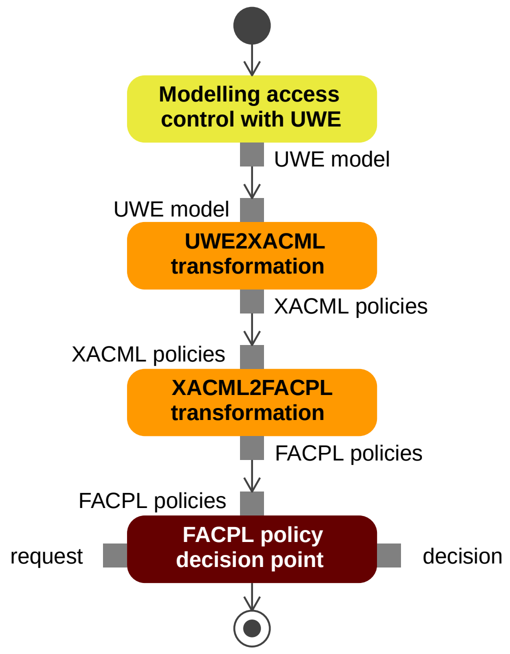


Figure 6.1: UWE2FACPL: toolchain

A request to access a resource can be created by, e.g., a remote-access gateway, a web server or an email user-agent. A XACML access requests is evaluated as follows: The authorization decision is made by the Policy Decision Point (PDP) by checking the *matching* between values of request's attributes and the corresponding values retrieved from the policies. The decision can be one among permit, deny, not-applicable and indeterminate: the first two values have an obvious meaning, while the third means that the PDP does not have any policy that applies to the request and the fourth means that the PDP is unable to evaluate the request.

Let us now consider the policy language provided by the standard. The basic element of this language is `Policy`. A `Policy` is composed of a `Target`, which identifies the set of credentials that the requestor must expose, and some `Rules`. Every `Rule` contains the logic for the access control decision and has an `Effect`, which can be either `Permit` or `Deny`. A `Policy` also specifies a combining algorithm that defines what is the final decision for a request when there are contradictory rule decisions (e.g. both permit and deny results are returned).

The most relevant algorithms are: `deny-overrides`, if any rule in the considered policy evaluates to deny, then the result of the policy is deny; `permit-overrides`, it is like `deny-overrides`, but permit takes precedence over the other results.

A `Target` is made of four sub-elements: `Subjects`, `Actions`, `Resources`, and `Environments`. Each category is composed of a set of target elements, each of which contains an attribute identifier, a value and a matching function. Such information is used to check whether the policy is applicable to a given request. Specifically, the matching function retrieves a value from the designed attribute in the request and matches it with the values specified in the target element, according to the function's semantics. If, for all four categories, at least a matching of a target element succeeds, then the policy is applicable to the request.

$$
\begin{aligned}
\mathit{Policies} &::= \{\mathit{Alg}\,;\mathsf{target}:\{\,[\,\mathit{Targets}\,]\,\}\,;\mathit{Policies}\}\\
&\quad\mid\ \langle \mathit{Alg}\,;\mathsf{target}:\{\,[\,\mathit{Targets}\,]\,\}\,;\mathsf{rules}:\{\mathit{Rules}\}\rangle\\
&\quad\mid\ \mathit{Policies}\quad \mathit{Policies}\\
\mathit{Alg} &::= \mathsf{deny\text{-}overrides}\ \mid\ \mathsf{permit\text{-}overrides}\ \mid\ \ldots\\
\mathit{Targets} &::= \mathit{MatchId}(\mathsf{value},\mathsf{name})\ \mid\ \mathit{Targets}\vee\mathit{Targets}\\
&\quad\mid\ \mathit{Targets}\wedge\mathit{Targets}\ \mid\ \mathit{Targets}\sqcap\mathit{Targets}\\
\mathit{MatchId} &::= \mathsf{string\text{-}equal}\ \mid\ \mathsf{integer\text{-}equal}\ \mid\ \ldots\\
\mathit{Rules} &::= (\mathit{Effect}\,[\,;\mathsf{target}:\{\mathit{Targets}\}\,][\,;\mathsf{condition}:\{\mathsf{expr}\}\,]\,)\\
&\quad\mid\ \mathit{Rules}\quad \mathit{Rules}\\
\mathit{Effect} &::= \mathsf{permit}\ \mid\ \mathsf{deny}
\end{aligned}
$$

Table 6.1: FACPL syntax

Besides the `Effect`, a `Rule` may specify a `Target`, which refines the applicability established by the target of the enclosing policy, and a `Condition`, i.e., a combination of functions that operate on values coming from the request. The `Effect` is propagated to the upper level policy if the `Target` of the rule matches and the `Condition` holds.

Policies can be combined into a `PolicySet`, which specifies a combining algorithm and a `Target`. The latter is evaluated before the targets of the included policies are.

The FACPL Policy Language

The Formal Access Control Policy Language (FACPL) [130] we describe in this section, which was originally written by the FACPL inventors and coauthors of [43], provides a manageable alternative syntax to XACML through a BNF-like grammar. FACPL syntax is reported in table 6.1. As usual, square brackets are used to indicate optional items.

To base an authorization decision on some characteristics of the request, like, e.g., the subject's identity or the resource's identifier, FACPL provides *(structured) names*, ranged over by name. They permit to identify specific values (called *attribute* values) contained in the request. The language is also equipped with *expressions* that permit to specify conditions.

FACPL policies can be simple policies of the form $\langle \mathit{Alg}\,;\mathsf{target}:\{\,[\mathit{Targets}]\,\}\,;\mathsf{rules}:\{\mathit{Rules}\}\rangle$ or, recursively, *policy sets* of the form $\{\mathit{Alg};\mathsf{target}:\{\,[\mathit{Targets}]\,\}\,;\ \mathit{Policies}\}$. Both policies and policy sets specify the algorithm for combining the results of the evaluation of the contained elements and a target to which the policy/policy set applies.

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

To obtain a more compact notation, FACPL represents a target as an expression built from *match elements*, i.e., terms of the form $\mathit{MatchId}(\mathsf{value},\mathsf{name})$, by exploiting an operator for logical disjunction, $\vee$, and two operators for logical conjunction, $\wedge$ and $\sqcap$. Each match element spells out a specific value that the subject/resource/action/environment in the decision request (identified by the given name) must match, according to the matching function *MatchId*. A disciplined use of structured names and the three logical operators permits properly expressing XACML targets. For further details on this topic, the reader is referred to [130].

A single policy contains a (non-empty) set of rules such as
(*Effect* [; target :{ $Targets$ }][; condition :{expr}]), each specifying: (i) an *effect*, which indicates the rule-writer's intended consequence of a positive evaluation for the rule (the allowed values are permit and deny), (ii) a *rule target*, which refines the applicability established by the target of the enclosing policy, and (iii) a *condition*, which is a Boolean expression that may further refine the applicability of the rule. In a rule, target and condition may be absent.

Regarding works on XACML's formalization, a largely followed approach is based on "transformational" semantics (see, e.g., [118, 35, 34]). The target formalisms have in their turn their own semantics. This makes it more difficult to understand the formal meaning of policies with respect to FACPL formal semantics, which directly associates mathematical objects (i.e., 4-tuples of request sets) to policies. These concepts are easier and more understandable than terms like, e.g., description logic expressions. In fact, FACPL semantics has been conveniently exploited to drive a formal-based XACML implementation. It differs from many XACML implementations (see, e.g., the OASIS website[15]), because it enables the development of reasoning tools. Besides, when policies do not change frequently, the FACPL implementation enables a faster decision because it does not need to parse the same XML tree at each request, but instead instantiates a Java object already in the classpath. In more dynamic scenarios, however, the generation of the PDP may add a constant time to policy evaluation. Finally, the use of a non-XML syntax for XACML is not new; e.g., a syntax similar to that of FACPL is proposed in [158], while a 'display' notation that combines a graphical interface with a natural language like format is introduced in [207]. However, such approaches do not rely on a formal semantics.

6.2.2 Policy Transformation

In this section, we describe our approach for generating XACML and FACPL policies from UWE models. As an example, we use our Hospital Information System case study (cf. section 5.1.3). The SDE is used to connect and execute all tools' methods that are presented in this section to a toolchain.

The focus of interest in HospInfo's Content model, which is depicted in figure 6.2, is on the `Patient` class with attributes like name, address, ward or gender. The classes `User` and `Role` (from the UWE User model) are included as well in figure 6.2, for showing the associations to the Content model elements. The user's attribute `id` is added to clarify that identity is equivalent to having the same identifiers.

Figure 6.3 depicts the Basic Rights model of HospInfo with access specifications for the classes `User` and `Patient`. The rule that administrators cannot change their own user account is depicted with the OCL [151] *authorizationConstraint* in the center of the UML diagram. Thereby, the variable `caller` stands for the operating user, i.e. the ≪updateAll≫ dependency between `Receptionist` and `Patient` specifies that the updates on all other attributes of `Patient` are permitted. Conversely, physicians can update all `Patient` attributes without any {except} restrictions.

Figure 6.4 shows a simplified excerpt of the main navigation state diagram for HospInfo. Basically, HospInfo consists of the two navigation areas depicted in figure 6.4: a visitor area (on the left) and an internal area (on the right), which is guarded according to the existing roles.

[15] Available XACML Implementations.
https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml#other

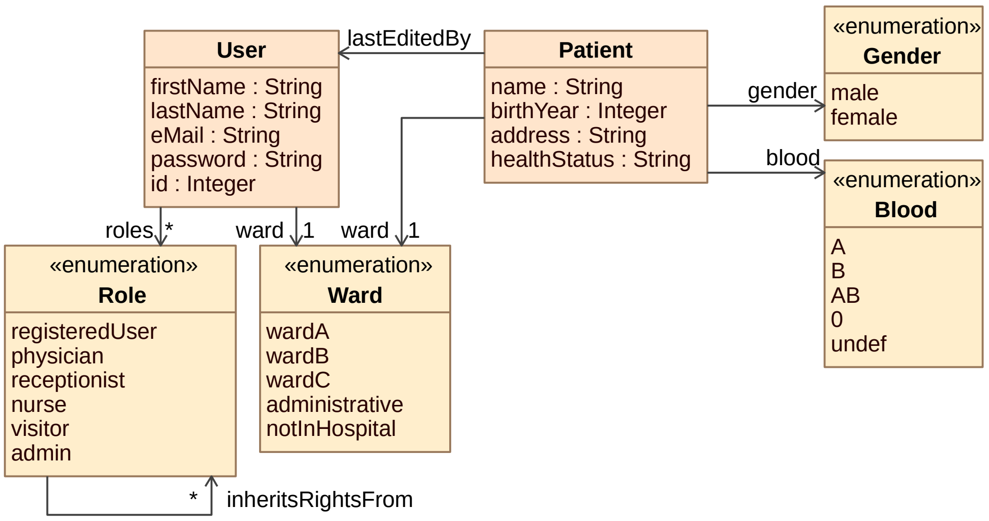


Figure 6.2: HospInfo Content model

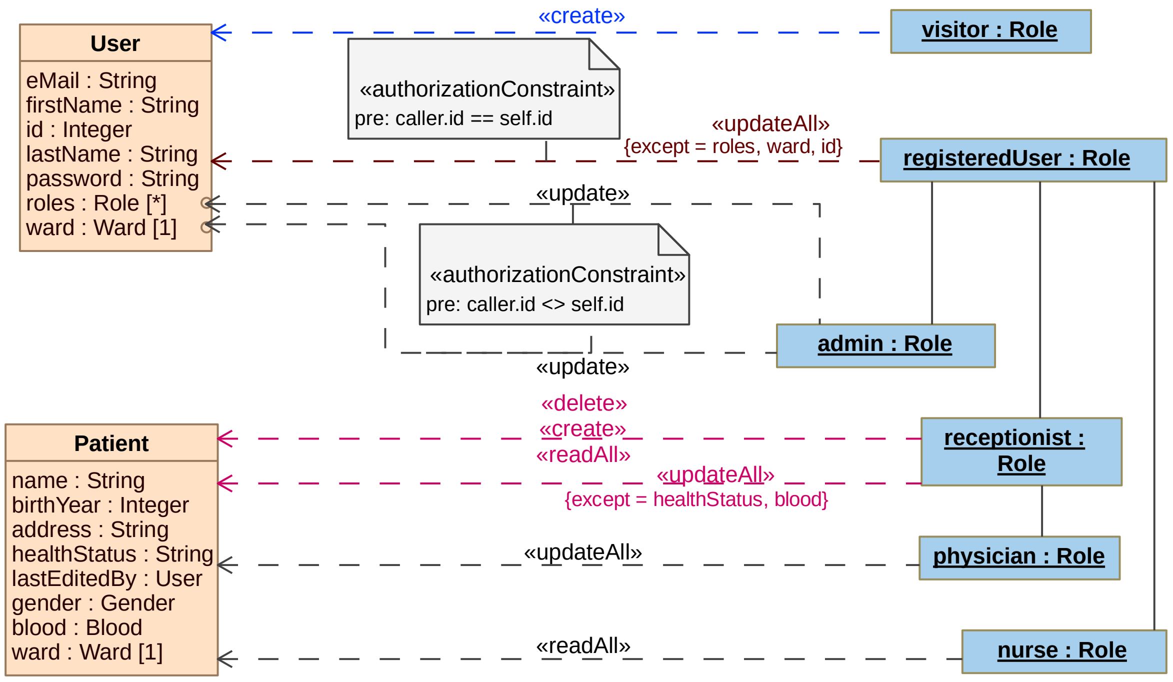


Figure 6.3: HospInfo Basic Rights model

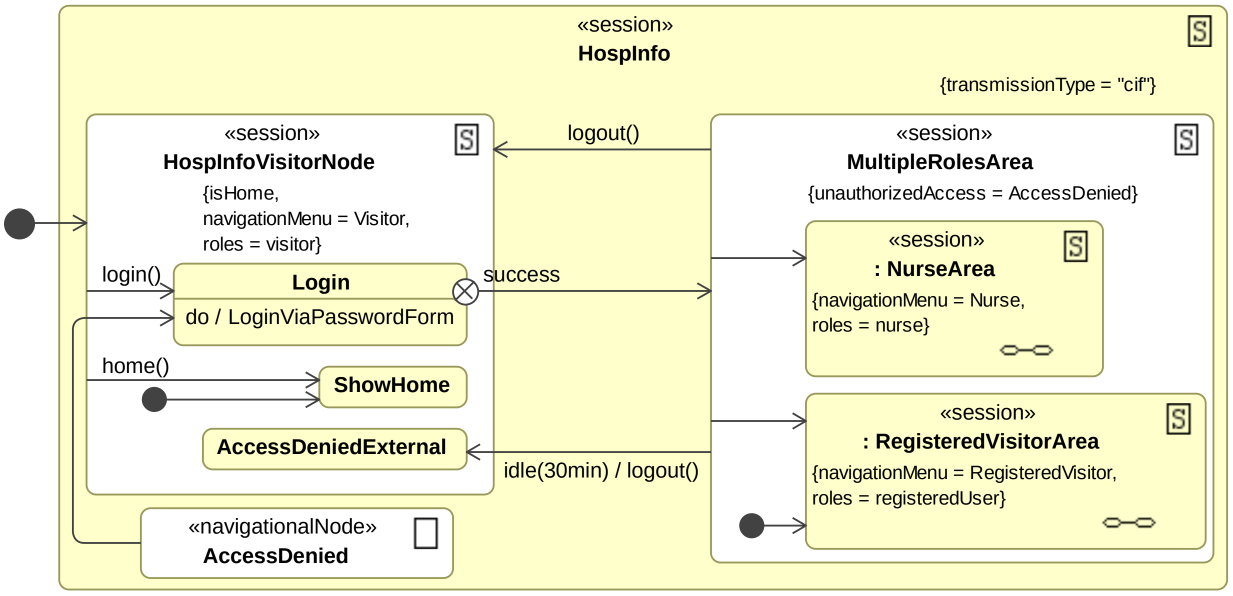


Figure 6.4: HospInfo navigation states (excerpt)

UWE to XACML

Technically, we extend MagicUWE to export XACML policies from UWE models. After transforming our HospInfo example to XACML policies, the length of the resulting XACML file exceeds several hundred lines. Thus, we sketch the rough structure of the file below.

```
PolicySet ..................... root element, permit-overrides
└─PolicySet ................... for each role
  ├─Target .................... role and all sub-roles
  └─Policy .................... for each pair of role and target
    ├─Target .................. constrained element
    └─Rule .................... permission for each action
      ├─Resources ............. e.g. attributes of a class
      ├─Actions ............... permitted action
      └─Condition ............. transformed constraint
└─Policy ...................... deny all
```

Intuitively, the transformation generates a `PolicySet` for each role, each of which contains one `Policy` for any class connected to the considered role (e.g. both `PolicySet` for `Receptionist` and `Physician` includes one policy for the content class `Patient`). Furthermore, a single `Policy` is used to deny access to all resources not specified in the `PolicySet`, which is the default behavior of UWE's Basic Rights models.

To allow a sub-role of a given role to use the permission specified by the super-role, the target of the `PolicySet` corresponding to the super-role is extended to also match requests from the sub-role (e.g. the target of the `PolicySet` for `receptionist` specifies two subjects, with roles `receptionist` and `physician`, respectively).

Each `Policy` for a constrained class contains one `Rule` for each action between the role and the class. For example, the `receptionist` policy comprises rules for actions ≪delete≫, ≪create≫, ≪read≫ and ≪updateAll≫ {except = healthStatus, blood}. Attributes targeted by `*All` actions are divided into a set of `Resources`, omitting those from the {except} tag. OCL constraints inside UML comments with ≪authorizationConstraint≫ stereotype are transformed to a `Condition`. The condition is located within a `Rule` representing the appropriate action. For the time being, we implemented only a few basic OCL constraints.

Technically, our UWE2XACML transformation from UML projects modeled with the UWE profile v.2.2 to XACML 2.0 is implemented using the modeling framework of Eclipse Juno[16]. We also used Xpand 1.2.1[17], a language specialized on code generation based on models defined by the modeling component of Eclipse. Xpand is based on workflows, which apply templates in order to parse the model and to produce the desired code.

To be able to use the project files of MagicDraw 16.8 for the transformation with Xpand, they have to be exported as Eclipse UML 2 (v3.x) XMI file. For complex tasks as the transformation to XACML, Java extensions are used from within the Xpand templates, because Java enables us, e.g., to group several dependencies with equal constraints to only one `Rule`. This is also needed for our HospInfo example regarding both update permissions (roles and ward) from administrators.

Our algorithm transforms not only the Basic Rights model, but also states with a {roles} tag from the Navigation model (see figure 6.4). The aim is to constrain whether or not a user is allowed to navigate to a certain area.

XACML to FACPL

The transformation, performed by the XACML2FACPL component (cf. section 6.2) loops over the policy sets creating the necessary data structures for the FACPL representation. The original XML document is read by using JAXB[18]. The loop over the elements is driven by the XACML schema definitions by traversing its data types. We show below the FACPL policies resulting from the transformation of the HospInfo Basic Rights model (only for receptionist and physician).

```
{permit-overrides ;
 target :{ string-equal(“physician”, subject.role) } ;
 ⟨permit-overrides ;
  target :{ string-equal(“patient”, resource.id) } ;
  rules :{(permit ;
        target :{ string-equal(“update”, action.id)
              ⊓ string-equal(“name”, resource.attr)
                ∨ string-equal(“birthYear”, resource.attr)
                ∨ . . .
                ∨ string-equal(“ward”, resource.attr) })
      (deny) } ⟩
 ⟨permit-overrides ; target :{ } ; rules :{(deny) } ⟩ }

{permit-overrides ; target :{string-equal(“receptionist”, subject.role)
                      ∨ string-equal(“physician”, subject.role)};
 . . .
```

[16]Eclipse. http://www.eclipse.org/
[17]Xpand. http://wiki.eclipse.org/Xpand
[18]JAXB. http://jaxb.java.net

```
 ...
⟨permit-overrides ;
  target :{ string-equal(“patient”, resource.id) } ;
  rules :{  ...
         (permit ; target :{ string-equal(“delete”, action.id) })
         ...
  } ⟩ }
```

6.2.3 Policy Evaluation

In this section, which was originally written by coauthors of [43], we sketch the formal semantics of FACPL, which is at the basis of the FACPL policy evaluation tool.

The semantics of FACPL policies is given in a denotational style, i.e., it is defined by a function $[\![\cdot]\!]_R$ that, given a policy/policySet and a set R of access requests, returns a *decision* tuple of the form

$$(\mathsf{permit}\!:\!R_p;\,\mathsf{deny}\!:\!R_d;\,\mathsf{not\text{-}applicable}\!:\!R_n;\,\mathsf{indeterminate}\!:\!R_i)$$

where R_p, R_d, R_n and R_i form a partition of R according to the results of the requests' evaluation. Notably, R can contain, e.g., all possible requests, only requests with a given structure or a single request. The definition of $[\![\cdot]\!]_R$ relies on an auxiliary function $(\!|\cdot|\!)_R$ that, given a target, returns a *matching* tuple of the form

$$(\mathsf{match}\!:\!R_m;\,\mathsf{no\text{-}match}\!:\!R_n;\,\mathsf{indeterminate}\!:\!R_i)$$

where R is partitioned into R_m, R_n and R_i according to the results of the target evaluation. We refer the interested reader to [130] for a full account of the FACPL semantics.

As an example, let us consider the following access requests:

```
request :{                              request :{
 (subject.lastName,“House”)              (subject.lastName,“Cameron”)
 (subject.role,“physician”)              (subject.role,“receptionist”)
 ...                                     ...
 (resource.id,“patient”)                 (resource.id,“patient”)
 (resource.attr,“healthStatus”)          (resource.attr,“healthStatus”)
 (action.id,“update”)                    (action.id,“update”)
}                                       }
```

The request on the left is made by the physician House for updating the health record of a patient, while the request on the right is made by the receptionist Cameron for performing the same action. Given the above requests and the HospInfo policies generated by the XACML2FACPL component of our toolchain, the FACPL semantics returns a decision tuple where the request on the left is in the permit set while the request on the right is in the deny set (indeed, receptionists have no permission to update patient health records).

The implementation[19] of the FACPL language is made in Java. The workflow of such a tool is graphically depicted in figure 6.5. It “compiles” a policy written in the syntax presented in

[19]Source and binary code of the FACPL implementation are available from http://rap.dsi.unifi.it/xacml_tools

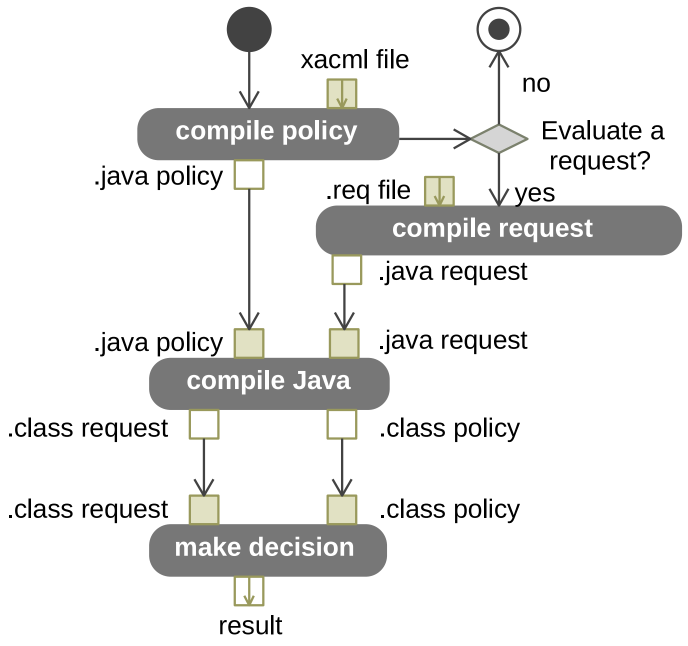


Figure 6.5: UWE2FACPL: FACPL Policy Decision Point

section 6.2.1 into a Java class following the semantics rules defined in [130]. Thus, a repository storing some policies, and the related PDP, consists of a Java archive containing all the Java classes generated from the policies. Similarly, an access request is compiled into a Java class. A policy decision is then computed by executing the generated policy code with the request code passed as parameter to an entry method. The generated PDP can then be integrated as a module into the code of the main web application.

6.3 ACT Toolchain: Testing Access Control Policies

In this section, which is based on joint work with Antonia Bertolino, Said Daoudagh, Francesca Lonetti, and Eda Marchetti [20], we aimed at automatically testing access control policies that are generated from UWE Basic Rights model. We therefore introduce an Access Control Testing toolchain (ACT) for designing and testing access control policies that includes the following features: (i) the graphical specification of an access control model and its translation into a XACML policy; (ii) the derivation of test cases and their execution against the XACML policy; (iii) the assessment of compliance between the XACML policy execution and the access control model. Thus, the execution of the full toolchain tests the tools in the toolchain themselves, which eases the replacement of tools. We illustrate the ACT toolchain using our SmartGrid Offers case study that has already been introduced in section 5.1.1.

6.3.1 Background

Background about XACML, the use of a PDP and the SDE can be found in section 6.2.1. We used the SDE to connect the tools that constitute the ACT Toolchain.

The complexity of the XACML language prevents the manual specification of a set of test cases capable of covering interesting critical situations or faults. This implies the need of automated generation of test cases. Martin presents the Targen tool [128] that derives the set of requests satisfying all possible combinations of truth values of the attribute id-value pairs found in the subject, resource, and action sections of each target included in the policy under test. Cirg [129] is able to exploit change-impact analysis for test cases generation starting from policies specification. In particular, it integrates the Margrave tool [84] which performs change-impact analysis so to reach high policy structural coverage. The X-CREATE tool [19, 18, 21] exploits the XACML Context schema defining the format of the test inputs, and also applies combinatorial approaches to the policy values. In [19] a comparison between X-CREATE and the tool Targen [129] has been performed in terms of fault-detection capability, and the results showed that X-CREATE has a similar or superior fault detection effectiveness and yields a higher expressiveness, as it can generate requests showing higher structural variability. [18, 21] present the advantages in terms of fault detection effectiveness of the testing strategies that are implemented into X-CREATE tool. Therefore, we selected X-CREATE to be part of our ACT toolchain.

6.3.2 The Access Control Testing Toolchain (ACT)

In this section we present the proposed Access Control Testing toolchain “ACT”, which includes the following main functionalities:

Model-driven Policy Design. The possibility to design a graphical specification of access control requirements and to convert the model into an XACML policy.

Test Case Generation and Execution. The selection of different testing strategies useful for deriving test cases and the possibility of executing them on the XACML policy.

Trace Analysis and Model Compliance. The analysis of test results and the consequent derivation of the *traces sets*, i.e. the execution of test cases on the XACML policy. The assessment of the compliance of the *traces sets* with the graphical access control model is also included.

For these steps, different tools are developed and integrated into the SDE. The SDE realizes the ACT Toolchain by executing functions of these tools in a row. Therefore, a function’s output can be graphically connected to another function input, as shown in figure 6.6. In this figure, the toolchain is depicted in the SDE’s graphical orchestrator.

As shown in figure 6.6, different activities are considered, each one involving a tool’s function available in the SDE. In the following, we list functions and briefly describe the corresponding services (tools) that implement them:

editProjectWithMagicUWE. MagicDraw[147] this is a modeling framework for specifying access control requirements, i.e. a graphical access control model, to simplify the designing of authorization constraints. MagicUWE is integrated as a plugin into MagicDraw.

transformUwe2xacml. UWE2XACML [43] provides an automatic translation of the graphical access control model into a XACML policy to avoid common errors and problems of manually written XACML policies.

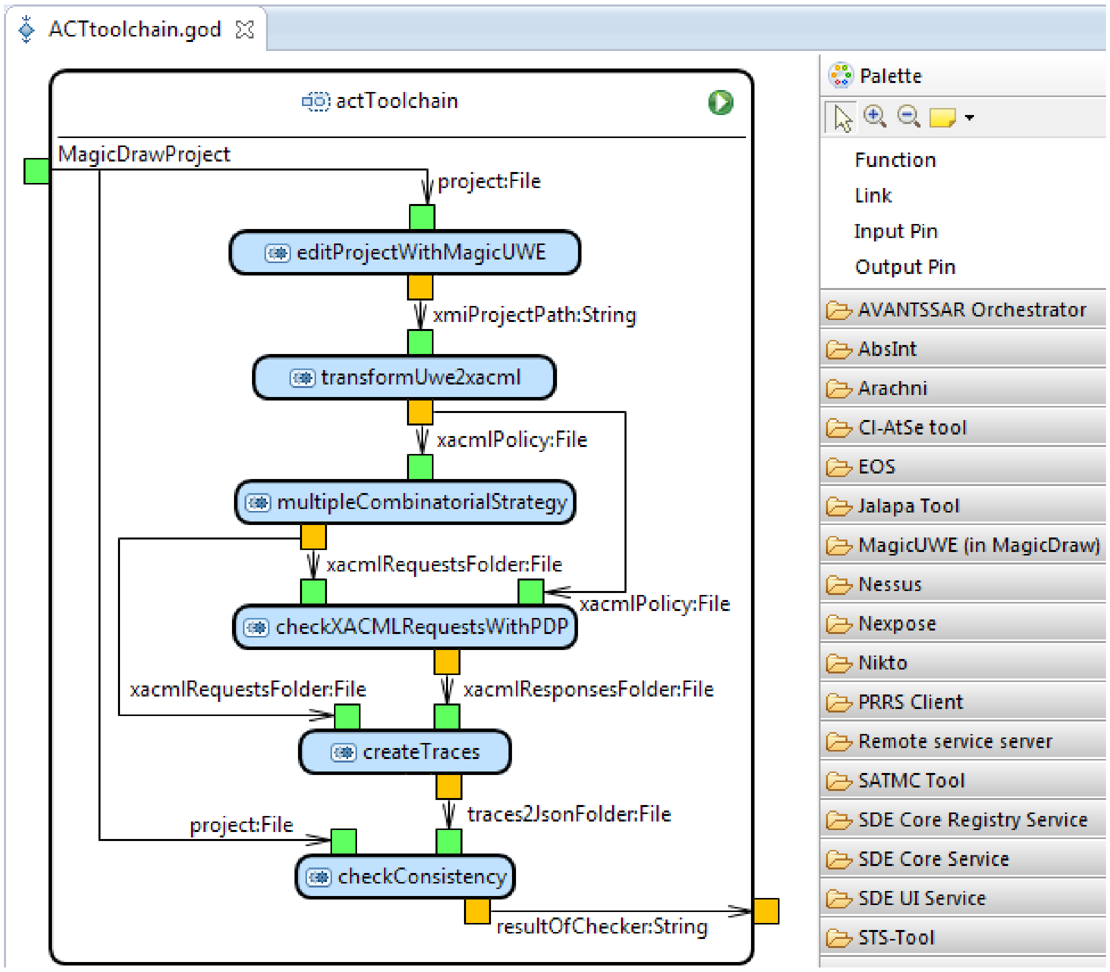


Figure 6.6: ACT Toolchain in the SDE: designing and testing access control policies

multipleCombinatiorialStrategy. The X-CREATE tool [19, 18, 21] enables automatic tests generation (i.e. XACML request generation) according to different testing strategies to speed up and improve the verification by reducing as much as possible time and effort due to test cases specification.

checkXACMLRequestWithPDP. The Sun PDP [209] is used to automatically execute test cases. Its output are XACML responses.

createTraces. The Trace Creator provides an automatic analysis of test results for deriving the model of the test execution called *traces sets*, followed by a transformation of the *traces sets* into sets of requests and responses expressed in the JSON format.

checkConsistency. MagicUWE offers this function (also called the "Checker") to automatically assess the compliance of the *traces sets* with the graphical access control model.

More technical details about the tools used for the implementation of the toolchain are provided in the following. Please note that X-CREATE's "multipleCombinatiorialStrategy", the functionality to "checkXACMLRequestWithPDP" and "createTraces" were created by coauthors of [43].

Model-driven Policy Design

The tool MagicDraw, with the plugin MagicUWE allows to graphically specify access control requirements whereas the tool UWE2XACML automatically transforms the derived model into a XACML policy.

MagicUWE. As has already been described in the previous section, MagicUWE supports the UWE notation and the UWE development process. For our toolchain, we use MagicUWE's support for modeling UWE diagrams. In particular, the toolchain first opens an existing UML project for modeling access control using the Basic Rights model of UWE. When the user finished modeling, he or she uses MagicUWE's menu to export the project as XMI [150]. In the same step, a link to the exported project is sent back to the SDE so that the toolchain can continue with the transformation to XACML.

UWE2XACML. The prototype UWE2XACML [43] is a tool for transforming role based access control policies modeled in the UML-based Web Engineering (UWE) language into XACML policies. As already described in section 6.2.1, UWE2XACML is written using the transformation language XPand [73] with Java extensions.

The input for UWE2XACML is an XMI-formatted UML model of an application modeled using the UWE profile. UWE2XACML iterates over the available roles while taking the role hierarchy into account. Additionally, the UML dependencies between the roles and the constrained elements are examined, i.e., the allowed actions are extracted. While iterating over the structure of the UML model, the XACML policy is written. In the end, it is formatted automatically according to the nesting of XML tags.

In our toolchain, UWE2XACML is immediately followed by X-CREATE for generating XACML requests for the newly generated XACML policy.

Test Case Generation and Execution

The automatic test cases generation and execution is implemented by means of the tool X-CREATE and the use of an access control system implementation called Sun PDP [209].

X-CREATE. The XACML policy derived by the tool UWE2XACML is used for deriving a set of test cases. For this we used the tool X-CREATE (XaCml REquests derivAtion for TEsting) [19, 18, 21], which implements different strategies for deriving XACML requests from a XACML policy. These strategies are based on combinatorial analysis [58] of the values specified in the XACML policy with the aim of testing policy evaluation engines and access control policies.

In the toolchain implementation, we decided to use X-CREATE's *Multiple Combinatorial* test strategy, because it provides a good compromise between test effectiveness and cost reduction [22]. In particular, for each policy, four sets are generated, the SubjectSet, the ResourceSet, the ActionSet, and the EnvironmentSet, containing the values of elements and attributes of the subjects, resources, actions and environments respectively. Random entities are also included in each set so that the resulting test plan could be used also for robustness and negative testing purposes. The entities are then combined to derive the XACML requests.

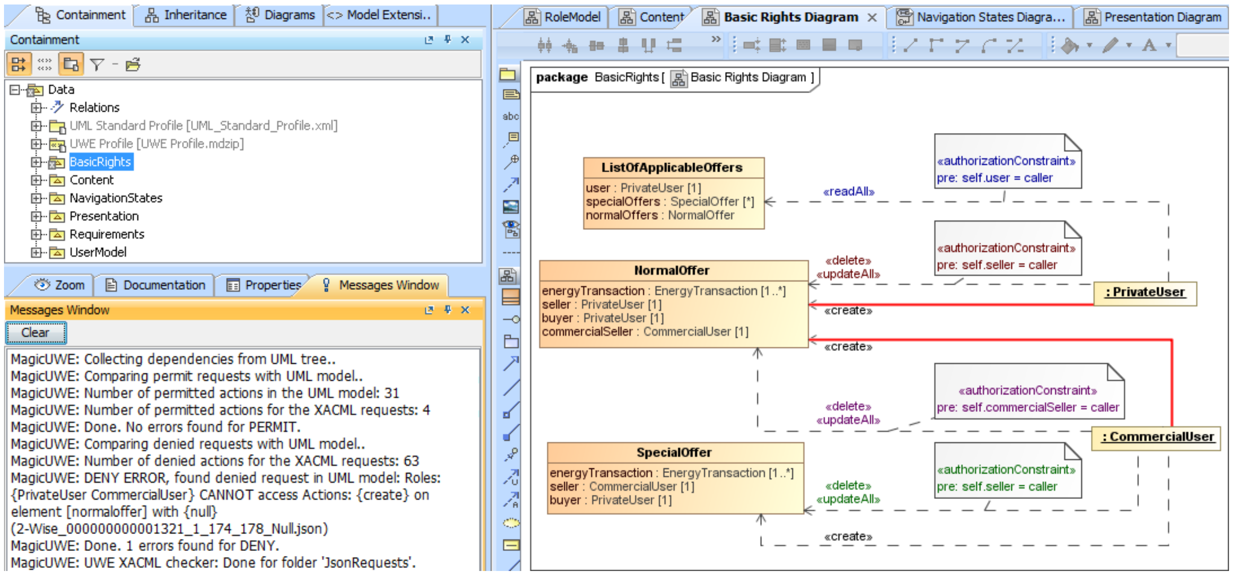
However, to avoid the possibility of an exponential cardinality of requests X-CREATE allows to fix the number of entities to be considered in each subset. Indeed, the necessary condition for a XACML request to be applicable on a field of the XACML policy (rule, target, condition) is that this request simultaneously includes all entities that are specified in that policy field. Thus X-CREATE exploits the minimum and maximum number of entities of the same type that have to be included in a request for reducing the set of generated test cases. The XACML requests are then generated by combining the subject, resource, action and environment subsets applying first a pair-wise, then a three-wise, and finally a four-wise combination, obtaining all possible combinations. In this process X-CREATE automatically eliminates duplicates. For more details, the interested reader is referred to [19, 18, 21].

Sun PDP. This component integrates a Policy Decision Point (PDP), which provides a support for parsing both policy and request/response documents, determining applicability of policies, and evaluating requests against policies giving the corresponding response (*Permit*, *Deny*, *NotApplicable* or *Indeterminate*). In ACT we included the Sun PDP [209], which is an open source implementation of the OASIS XACML standard, written in Java.

Results Analysis and Verdicts Generation

The XACML requests and the corresponding PDP responses are used to trace the policy execution and to assess the compliance of the derived policy with the graphical access control model.

Trace Creator. The Trace Creator gets as input the set of XACML requests together with the corresponding responses from the Sun PDP components and derives the *traces sets*. Different methodologies for classify the couples (request, response) are available, which rely on the opportune combination of the subjects, resources, actions, environments and corresponding responses. In our ACT Toolchain we consider the classification according to the PDP responses. Thus the couples (request, response) are divided into three groups: those having *Permit* as response (i.e.


Containment
Inheritance
Diagrams
<> Model Extensi..
Data
Relations
UML Standard Profile [UML_Standard_Profile.xml]
UWE Profile [UWE Profile.mdzip]
BasicRights
Content
NavigationStates
Presentation
Requirements
UserModel
Zoom
Documentation
Properties
Messages Window
Clear
RoleModel
Content
Basic Rights Diagram
Navigation States Diagra...
Presentation Diagram
package BasicRights [Basic Rights Diagram]
ListOfApplicableOffers
user : PrivateUser [1]
specialOffers : SpecialOffer [*]
normalOffers : NormalOffer
NormalOffer
energyTransaction : EnergyTransaction [1..*]
seller : PrivateUser [1]
buyer : PrivateUser [1]
commercialSeller : CommercialUser [1]
SpecialOffer
energyTransaction : EnergyTransaction [1..*]
seller : CommercialUser [1]
buyer : PrivateUser [1]
«authorizationConstraint»
pre: self.user = caller
«authorizationConstraint»
pre: self.seller = caller
«authorizationConstraint»
pre: self.commercialSeller = caller
«authorizationConstraint»
pre: self.seller = caller
«readAll»
«delete»
«updateAll»
«create»
«create»
«delete»
«updateAll»
«delete»
«updateAll»
«create»
: PrivateUser
: CommercialUser

```
MagicUWE: Collecting dependencies from UML tree..
MagicUWE: Comparing permit requests with UML model..
MagicUWE: Number of permitted actions in the UML model: 31
MagicUWE: Number of permitted actions for the XACML requests: 4
MagicUWE: Done. No errors found for PERMIT.
MagicUWE: Comparing denied requests with UML model..
MagicUWE: Number of denied actions for the XACML requests: 63
MagicUWE: DENY ERROR, found denied request in UML model: Roles:
{PrivateUser CommercialUser} CANNOT access Actions: {create} on
element [normaloffer] with {null}
(2-Wise_000000000001321_1_174_178_Null.json)
MagicUWE: Done. 1 errors found for DENY.
MagicUWE: UWE XACML checker: Done for folder 'JsonRequests'.
```

Figure 6.7: ACT Toolchain: output of the MagicUWE Checker

the *Permit set*), those having *Deny* as response (i.e. the *Deny set*), and those having either *NotApplicable* or *Indeterminate* (the *Other set*) as response. To simplify the validation process the elements of the *traces sets* are translated into Java Script Object Notation (JSON)-formatted couples. The choice of the JSON notation instead of XML was made because JSON is a lightweight format that is easier to read and parse than XML[20].

MagicUWE Checker. The final element of our toolchain is the MagicUWE Checker, which checks that the elements of the *traces sets* comply with the original graphical access control model, i.e. UWE's Basic Rights model.

The main function of the MagicUWE Checker is `checkConsistency`, which takes two arguments: the UML project and the JSON-formatted traces sets. The Checker's functionality is implemented as a part of MagicUWE. This means that the first and the last element of our toolchain are functions that execute MagicDraw with the plugin MagicUWE installed.

The Checker tests if each couple (request, *Permit*) of the *Permit set* can be associated to stereotyped dependencies between roles and concepts expressed in the model. In particular, if a request contains a user that has more than one role, the Checker verifies if there exists at least one stereotyped dependency between one of the mentioned roles and concepts expressed in the model. Additionally, each couple (request, *Deny*) is tested, to make sure that there is an action in the request which does *not* appear in the UWE model. If a user requested more than one action, the Checker verifies if at least one of these actions is not shown in the UWE model (i.e. denied in the model). The elements belonging to the *Other* set of traces are not considered.

As depicted in figure 6.7, MagicDraw provides three main windows: on the right, a UWE Basic Rights diagram is shown, which will be covered in our case study (cf. figure 6.9). On the top left, the element tree of the model is depicted, together with the root of the UML profile and UWE profile. On the lower left, the log of MagicUWE's Checker provides details of the requests

[20]Comparisons of JSON and XML. http://www.json.org/xml.html

under test. There, it is indicated if the PDP allows a request that is not permitted in the model or the PDP denied a request that is modeled in the Basic Rights diagram. In the latter case, the Checker also flags mistakenly denied dependencies in the graphical model. These bold, (red) dependencies and the Checker's log can then be used for debugging.

In the end, the log is automatically handed back to the SDE where it is shown as result of the ACT Toolchain. In the future, new tools could extend our toolchain, as e.g., a dashboard tool that parses the log to show a green light if the Checker reported no errors.

Toolchain Integration

For the assembly of our toolchain, we integrated the tools in the SDE, i.e., we created a SDE wrapper for tools that were not yet integrated. Finally, we connected function of these tools to a toolchain using the SDE's graphical Orchestrator. As depicted in figure 6.6, the SDE Orchestrator is easy to use: we created a new *Function*, called "actToolchain" using the *Palette* on the right hand side. Using drag & drop, we added an input and an output pin and the tools' functions (e.g. MagicUWE: checkConsistency). After connecting inputs and outputs with *Link*s, the toolchain can be executed using the green play-button on the upper right corner. At the moment, there is a minor drawback, as plugins based on XPand, such as UWE2XACML, only function when they are executed in the Eclipse development mode.

6.3.3 Case Study

In this section we apply the ACT Toolchain on the SmartGrid Offers application, which allows offers to be created and bought, as already described in section 5.1.1.

Step 1: modeling access control. In the first step of our toolchain, we use MagicDraw with MagicUWE installed to model the classes and their associations, as shown in figure 6.8. It is noticeable that `Role` and its subclasses are located in UWE's Role model and are just shown in the Content diagram in order to present the connection between roles and content.

Figure 6.9 depicts the Basic Rights diagram of our case study. For example, a user with the role `PrivateUser` is allowed to read all attributes of an instance of `ListOfApplicableOffers`, which refers to the customized list of energy offers that is presented to a user.

However, nobody should be allowed to access a list of someone else, which is expressed using the OCL [151] authorization constraint: "pre: self.user = caller". The term `self` is defined as a referrer to the current class, `user` is an attribute of `ListOfApplicableOffers` (cf. figure 6.8) and `caller` refers to the user which is executing an action.

Step 2: transforming the UWE model to a XACML policy. After the UWE model is exported as XMI, the tool UWE2XACML transforms it to a XACML policy.

According to the description in section 6.2, the transformation intuitively works as follows: It generates a XACML *PolicySet* for each role, each of which contains one *Policy* for any class connected to the considered role. Furthermore, a single *Policy* is used to deny access to all resources not specified in the *PolicySet*, which is the default behavior of UWE's Basic Rights model.

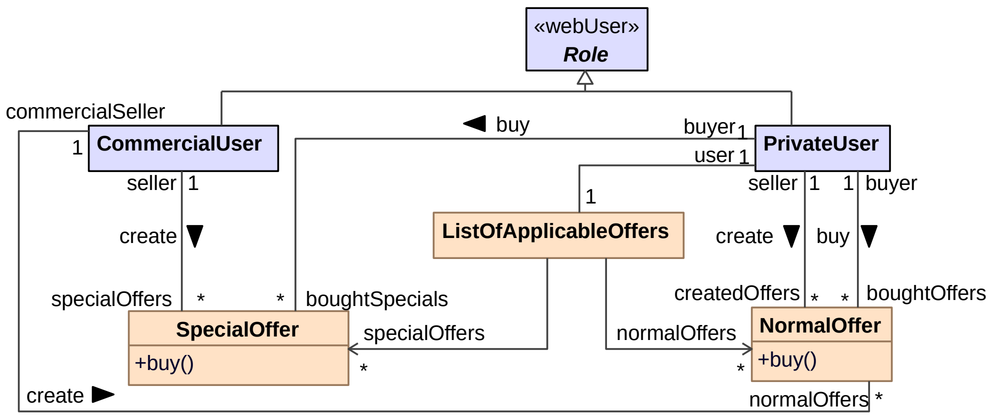


Figure 6.8: ACT Toolchain: SmartGrid Offers – Content model with roles from Role model

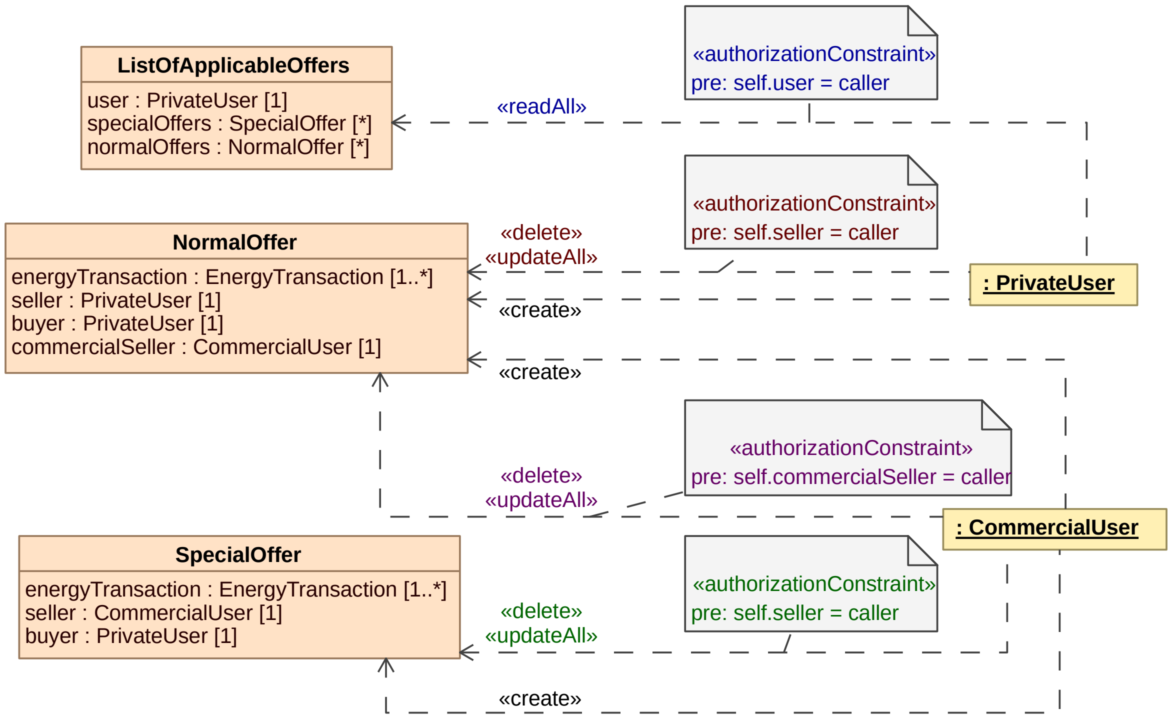

ListOfApplicableOffers
user : PrivateUser [1]
specialOffers : SpecialOffer [*]
normalOffers : NormalOffer [*]
«authorizationConstraint»
pre: self.user = caller
«readAll»
NormalOffer
energyTransaction : EnergyTransaction [1..*]
seller : PrivateUser [1]
buyer : PrivateUser [1]
commercialSeller : CommercialUser [1]
«delete»
«updateAll»
«authorizationConstraint»
pre: self.seller = caller
: PrivateUser
«create»
«create»
«delete»
«updateAll»
«authorizationConstraint»
pre: self.commercialSeller = caller
: CommercialUser
SpecialOffer
energyTransaction : EnergyTransaction [1..*]
seller : CommercialUser [1]
buyer : PrivateUser [1]
«delete»
«updateAll»
«authorizationConstraint»
pre: self.seller = caller
«create»

Figure 6.9: ACT Toolchain: SmartGrid Offers – Basic Rights model

XACML policies generated by UWE2XACML are structured as follows (in brackets we show the policy for the uppermost ≪readAll≫ dependency in figure 6.9, which connects `PrivateUser` and the class `ListOfApplicableOffers`):

`PolicySet`	root element, permit-overrides
└─ `PolicySet`	for each role (e.g., contains what a PrivateUser is allowed to do)
└─ `Target`	role (and sub-roles if any) (e.g., PrivateUser)
└─ `Policy`	for each pair of role and target (e.g., for actions a PrivateUser can do on ListOfApplicableOffers)
└─ `Target`	constrained target element (e.g., listofapplicableoffers, as we use lower cased class names in the policy)
└─ `Rule`	permission for each action (e.g., concrete permission what to read)
└─ `Resources`	attributes of the target class (e.g., listofapplicableoffers.user, listofapplicableoffers.specialOffers, listofapplicableoffers.normalOffers)
└─ `Actions`	permitted action (e.g., read)
└─ `Condition`	transformed OCL constraints (e.g., we use the XACML function `integer-equal` to make sure that self.user.id equals caller.id). Besides, we restrict the number of actions and the number of classes per request to 1.
└─ `Policy`	default: deny all, which is not allowed explicitly

Step 3: generating requests for the XACML policy. Using the tool X-CREATE a set of XACML requests is derived starting from the XACML policy. A large number of requests are generated using the *Multiple Combinatorial* testing strategy provided by X-CREATE, which combines all the possible values of elements and attributes of the subjects, resources, actions and environments respectively. In the following, we present two generated request examples. In the first, a user with the role `PrivateUser` wants to create an instance of the class `NormalOffer`. In the second, a private user requests the creation of a `SpecialOffer`.

`Request`	root element
└─ `Subject`	the subject is PrivateUser
└─ `Resource`	the resource is NormalOffer
└─ `Action`	the action is `create`

`Request`	root element
└─ `Subject`	the subject is PrivateUser
└─ `Resource`	the resource is SpecialOffer
└─ `Action`	the action is `create`

Step 4: checking XACML requests on the PDP. The set of requests generated with X-CREATE are evaluated against the XACML policy on the Sun PDP and the corresponding

XACML responses containing the access results are collected. In our two cases, the responses are `Permit` for the first and `Deny` for the second request.

Step 5: creating traces. A request / response values based filter is applied for deriving the *traces sets*. Then the couples (request, response) are divided into three groups: the *Permit set*, the *Deny set*, and the *Other set*. The traces are then converted to JSON.

In the following, we give an example for a JSON-formatted trace.

```
{"XacmlRequest":{
  "Attributes":[],
  "InstanceIDs":{"self.seller.id":"94","self.caller.id":"94"},
  "Classes":["normaloffer"],
  "Decision":"Permit",
  "Actions":["delete"],
  "Roles":["PrivateUser"]}
}
```

Step 6: checking consistency of responses and initial model. The resulting differences between the XACML responses and the initial model are shown, if any. In case a faulty transformation tool in step 2 would have altered the XACML policy so that it is not allowed to create normal offers, the first request in step 3 (and all other requests for creating normal offers) would be denied in step 4. Thus, in step 6, the Checker can detect an inconsistency between the access control model and these requests, as, in the access control model of figure 6.9 users are authorized to create a normal offer. MagicUWE highlights the inconsistency with bold and red lines in the Basic Rights model, as depicted in figure 6.7. Consequently, the developer can start debugging by studying the Checker's log which points to the faulty trace file(s). Backtracking along the toolchain is possible, as the intermediate data that is handed over from one tool to the next is saved locally.

In practice, developers can also get the advantages of parts of the ACT toolchain. For example, already existing XACML policies can be modeled with UWE, as long as they are restricted to role based access control. This might help developers to regain an overview of their policy and to debug it. In this case, a part of our toolchain would be used, starting with the tool X-CREATE. UWE's Basic Rights model, which is needed by the Checker, would then be the new model of which the developers want to know if it is yet compliant to the XACML policy they provided. If it is not, the Checker's log would provide hints to missing or misplaced modeling elements.

6.4 ActionUWE: Transforming UWE to ActionGUI

Both, ActionGUI [11] and UWE 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. This section, which is an excerpt of joint work [38] with Miguel Ángel García de Dios (an expert for ActionGUI), sketches main ideas of a transformation, called "ActionUWE", which aims at semi-automatically generating secure code from high-level UWE models by transforming them into ActionGUI models.

Semi-automatically means that UWE models have to be enriched by certain elements before the transformation takes place and that the generated ActionGUI models have to be refined so that they contain all details needed for an application's actual execution.

In our technical report [38], we use our HospInfo case study to detail the transformation. However, in this section we just briefly present the main ideas of ActionUWE, without going into detail, as it turned out that implementing and using ActionUWE in practice (in order to generate code from ActionGUI models) would not be truly efficient. At least not when compared with traditional coding of the limited kind of applications that could be generated.

6.4.1 Background

As already discussed in section 2.3.2, most modeling approaches do not support modeling security features, whereas the UWE approach and the ActionGUI approach by Basin et al. [12] define models for security mechanisms like access control. In ActionGUI, the whole application logic is represented using OCL, which allows to generate complete web applications. ActionGUI's proprietary notation comprises the following models:

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.

No model-driven solution for secure web applications exists that unite the advantages of UWE and ActionGUI: On the one hand, the advantages of the high-level of abstraction of UWE with its many views (separation of concerns) that help practitioners to model an application's architecture along with its security features. Thus, developers immediately get an overview of requirements that have been elicited and design decisions that have already been taken. On the other hand, the advantages of a modeling language like ActionGUI which is based on a formal specification of the whole application logic and its access control policies. For basic user interfaces, these policies allow the generation of secure web applications where the security policies are automatically embedded in the user interface, which means that elements are only shown if the user is allowed to read the underlying data or to execute the functionality behind.

6.4.2 Transformation

Our model-driven approach ActionUWE combines the approaches UWE and ActionGUI by a transformation. Before the transformation, the UWE model has to be redefined to connect views, e.g., by linking elements of the UWE's Presentation model to states and transitions (representing method calls) of the Navigation model. Therefore, we added new UML tags like {navState} or {navTrans} to presentation elements in the UWE profile that can point to concrete states or transitions.

The ActionUWE transformation itself is executed in four steps:

Step 1 initializes the ActionGUI model and transforms available UWE menus to ActionGUI `Menu` classes.

Step 2 adds further information of the Presentation model to ActionGUI.

Step 3 transforms the UWE Navigation model to ActionGUI without regarding security features.

Step 4 translates and adds the Role-Based Access Control (RBAC) constraints and the navigational access control features to ActionGUI.

Afterwards, validation checks that are available for ActionGUI models can be used to examine the model before it is subjected to ActionGUI's model-to-code transformation.

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 Presentation 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 Navigation model is mapped to certain `Action` and `Event` elements of the ActionGUI model.

To make the transformation possible, we had to slightly extend the metamodel of ActionGUI and the profile of UWE and to specify additional preconditions for the UWE models, as ActionGUI models cannot represent all kinds of web applications. An example for such a precondition is that we assumed that one menu exists and this menu changes exactly one other panel. In future versions of ActionGUI this restriction might disappear, but this would lead to a rather complex way of describing which panel (or subordinated panel) should be exchanged at runtime. For a more detailed description of the ActionUWE transformation the interested reader is referred to our technical report [38], which can be found online.

6.5 SNPs: Modeling, Testing and Securing Navigation Flow

Although robust solutions for managing authentication, authorization and session management in web applications exist, the question of effectively controlling the navigation flow for different users remains challenging. In this section, which is based on joint work with Martín Ochoa and Roman Schwienbacher [47], we propose a methodology that allows testing and enforcing Secure Navigation Paths (SNPs) that were modeled with UWE. We focus on the integrity of the navigation paths as intended by the application owner, i.e., the order in which authorized resources of an application should be accessed by a given user. In addition, we automatically generate a server-side monitor enforcing such policies and discuss how models can be used to generate tests in case a monitor is absent. Finally, we report on tool support for this methodology and apply it to the SmartGrid Bonus case study we have already introduced in section 5.1.1.

6.5.1 Background

In this section we briefly recall the addressed challenge of enforcing secure navigation paths and approaches related to business workflow integrity.

Among the most challenging web application vulnerabilities are the ones involving the misuse of the application logic itself. As stated by the Common Weakness Enumeration (CWE) [60]:

> Errors in business logic can be devastating to an entire application. They can be difficult to find automatically, since they typically involve legitimate use of the application's functionality.

Exploiting flaws in the business workflow is a common attack to the application logic. These exploits typically consist of jumping to certain URLs, bypassing critical controls of an intended flow or manipulating the parameters of legal requests. The consequences of those attacks can be diverse: bypassing log-in controls result in authentication breaches whereas skipping certain controls in a trading operation might result in monetary benefits to the attacker. Examples of documented vulnerabilities in popular web applications include the Yahoo SEM Logic Flaw [99]: if one deposited USD \$30 into an advertising account, Yahoo would then add an additional USD \$50 to that account. The sign-up process was able to be circumvented such that failing to deposit the USD \$30 still allowed to receive the additional USD \$50. Other examples include bypassing of age restrictions in Youtube, access to private photos in MySpace (resulting in attacks to celebrities) among others (see [8]).

In recent years, much attention has been given to validating user input to web applications to prevent code-injection (i.e. SQL, XSS), but very few tools and methodologies are available to prevent and test logical errors.

Braun et al. [31] published a robust approach for SNPs for MVC-based web applications where policies are specified using an ad-hoc textual notation. They also tackle race-conditions and handling of multiple tabs within a browser, which is currently outside of the scope of our approach. The parameter constraint in our approach was partially inspired by their textual policy language, although our approach is mainly based on web pages, not on methods. In UWE, some problems are inherently solved, as e.g., superstates exist so that all transitions can be easily specified and there is no need to invent extra notations for the ability to change decisions later or for the availability of the back button.

In 2002, Scott et al. [192] described a system which is also based on a solution using a monitor. A textual policy specifies validation constraints, mainly for parameters and cookies, in a language called Security Policy Description Language (SPDL). This policy is then compiled to code which is executed by the monitor when a page is accessed. Additionally, Message Authentication Codes (MACs) can be added by the monitor when delivering a page so that, e.g., hidden form fields can be secured from changes at the client-side.

Halle et al. [98] define a navigation state machine with session traces with a focus on a formal model. However, the state machine is only a simple one with no further information than a sequence of states which does not include parallel states. If desired, their formal approach might be extended to describe UWE's Navigation model, including information given by the stereotypes, tags and parallel states.

In [140] a method for secure design of business application logic is sketched. It comprises strategies such as analyzing weaknesses caused by misconfiguration of server-side components or by errors in the application logic. They suggest to test several kinds of parameters, however, they do not provide tool support. Furthermore, it is recommended to define a clear design of the architecture, especially for components which update session data. The authors aim to provide a good practice which certainly can be combined with our approach.

Additionally, a tool for servlet-based web applications is provided by Felmetsger et al. [81]. The tool, called *Waler* uses a composition of dynamic analysis and symbolic model checking: Regarding the dynamic analysis, it observes the normal operation of a web application in order to infer behavioral specifications that are filtered to reduce false positives. Afterwards, symbolic model checking is used to identify program paths that are likely to violate these specifications. Compared to our approach, models do not have to be created manually, which is convenient, especially for legacy applications. However, the price is that flaws in the navigation paths can only be detected with a certain possibility. SPaCiTE [36] is a tool that generates concrete attack tests based on model checking and mutation operators. It has been applied so far for testing RBAC and XSS, but not for business workflow integrity.

6.5.2 Modeling Approach

We focus on how to model basic SNPs and on a notation for specifying parameters for web pages.

Basic SNPs

As introduced in section 5.2, UWE provides several views. For each view, an appropriate UML diagram is selected. In UWE, UML state machines are used to model the navigation structure of a web application in the Navigation model. In our case, states correspond to navigational nodes that are implemented as web pages. Transitions define all possibilities to navigate from one page to another and thus specify a policy for the navigation. By using the UWE approach, it is also possible to annotate transitions and states with necessary permissions required to access a resource.

Building on UWE's Navigation model, we define SNPs as follows (cf. figure 5.11): In case a transition leads from state A to B, a user can visit page B after having visited page A. If it should be possible to go back (e.g. with the back-button in the browser), another transition has to connect B with A. For more than one option, an arbitrary number of transitions can be used.

We use the behavior of UML composite states to model links which should be available within a certain area at any time. "Composite" means states can be nested within a composite state. If state Y and Z and an initial node are nested inside a state A and the initial node is connected to Y, then transitions to A activate Y (because of the inner initial node). A transition that leads from A to an arbitrary new state B can be fired from inside A, no matter if Y or Z is active. Consequently, A does not correspond to a web page itself, but groups others. An example of a UWE Navigation diagram expressing SNPs is depicted in our case study. In this way we model SNPs at a high level of abstraction, as no technical details have to be given.

We also use the Navigation model of UWE to specify navigational access control (cf. section 5.3). For each «session» stereotype, denoting a user's session, a tag called {roles} can point to a set of roles from UWE's Role model. In order to be able to access the web page represented by a state, a user has to have at least one of the roles that are allowed to access this state. If this is not the case, the tag {unauthorizedAccess} specifies which state should be used instead. This state can then represent, e.g., a page with an error message or an advertisement for a more expensive account.

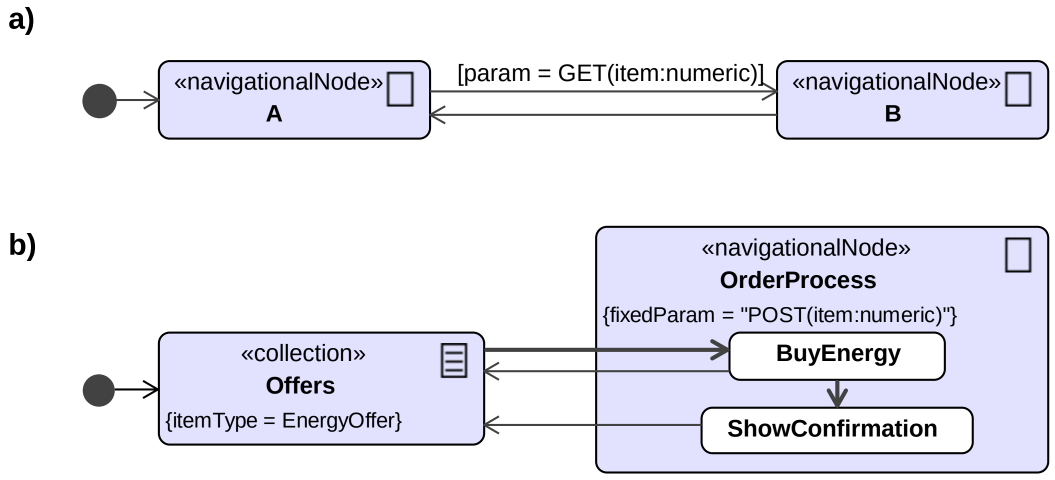


Figure 6.10: SNPs: expressing SNPs with UWE's Navigation model

SNPs with Checked Parameters

However, many web pages use parameters to pass, e.g., user input or session IDs to the next page. To model allowed parameters specifically, we extend UWE's Navigation model so that a minimum of technical information can be specified, if needed. Our extension is inspired by Braun et al. [31], who came up with a textual control-flow definition language for Model-View-Controller (MVC)-based web applications. As our approach does not specify method names, but page names, it is not restricted to MVC-based applications. Furthermore, information about allowed parameters can easily be added to existing Navigation diagrams, so that related information about authentication, secure connections, navigational control as well as SNPs can be conceived at a glance. We add parameters to transitions using a guard like [param = GET(par1:type1, par2:type2)]. GET or POST are allowed and types can be Boolean, numeric or string (cf. part *a* of figure 6.10).

Sometimes, a parameter should be added to requests within a certain area, using the same value as at the first occurrence. For instance, a session Identifier (ID) is not allowed to change during a session and selected items should not change during the payment process. This can be modeled by a composite state which comprises all transitions that should use fixed parameters. All navigational states in UWE inherit from the stereotype ≪navigationalState≫, for which a tag called {fixedParam = POST(par1:type1, par2:type2)} can be set. Global parameters are applied to all transitions where the target state is located within the composite state. The choice of GET / POST for the composite state and affected transitions has to be coherent in case inner transitions specify further parameters. When leaving and entering the composite state again, the values of the fixed parameters can of course be different than before.

Part *b* of figure 6.10 depicts this behavior: the bold transitions inherit the fixed parameter. This means the value of {item} is set by the transition targeting `BuyEnergy`. Afterwards, it cannot be changed until the `OrderProcess` is left. The stereotype ≪collection≫ denotes that several `Offers` are shown. Each offer is of the type `EnergyOffer`, which is defined by the tag {itemType}. If an offer is bought, the confirmation cannot be shown for another one.

For simplicity, in the remainder of this section we focus on modeling and enforcing SNPs on web applications without imposing constraints on the parameters. This is reasonable since impor-

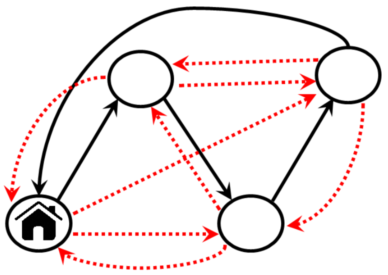


Figure 6.11: SNPs: test generation

tant security parameters such as the session ID are already handled automatically by development frameworks.

6.5.3 Testing

In this subsection, we describe how to generate navigational test cases for web applications with access control policies regarding SNPs. The main goal is to cover every possible navigation context considering navigation history, current navigational node, user role and access permission result. Comparing these results with given access control policies, we can detect possible access control misbehavior issues.

In order to build navigational test cases we use the following approach based on an UWE SNP policy given input: for each user role we walk through every available SNP starting at the entry node which is marked as {isHome}. We navigate from node to node inside the SNP until we come back to an already visited node. This way we can test if the application behaves according to the specification (we can identify false negative access control behavior by collecting occurring access denials). In order to detect possible violations of the integrity of the policy, we simply try to leave the SNP on every node by requesting each node which is currently not accessible by an outgoing transition and thus not allowed. In this context, every granted access represents an incorrect behavior. Figure 6.11 depicts such a navigation through a SNP including illicit node requests on every node.

However, with every violating request we cause a navigation history which does not correspond to the original SNP, and we possibly harm the state of the application. Therefore, we need to reconstruct the previous SNP-valid navigation context to go ahead: First, we have to fall back to the current entry node to clear the navigation history. Second, we have to repeat the navigation progress on the SNP until we achieve the previously visited node inside the SNP to reconstruct the navigation context.

This testing process is efficient, since it has a complexity of $\mathcal{O}(rn^3)$. Assuming there are n possible navigational nodes and r user roles: Every navigational node has up to $n-1$ neighbors which are not accessible by an outgoing transition. By testing all roles, we get an amount of $rn(n-1)$ test cases which gives an upper bound of $\mathcal{O}(rn^2)$ tests. Considering the backtracking behavior to reconstruct the navigation context we have to visit up to $n-1$ additional nodes for every forbidden node. Consequently, we get a final complexity of $\mathcal{O}(rn^3)$. However, testing parameters cannot be exhaustive, as parameters can contain arbitrary values. Extending our approach to consider constraints on parameters is thus left as future work.

6.5.4 Tool Support

This subsection presents tool support that we developed to validate our approach. In a bachelor's thesis, supervised by the author, Schwienbacher [190] implemented *MagicSNP* to export navigational access control rules that can be enforced by our *SNPmonitor*. For tests, our *SNPpolicyTester* is employed.

MagicSNP. In order to validate a Navigation model and moreover to extract the corresponding access control semantics we developed a CASE tool plugin for MagicDraw called *MagicSNP*. By iterating through all hierarchical states and analyzing incoming transitions, state names and tags, our tool fetches relevant information about navigational access control and SNPs. The JSON-structured result can be taken as input for a security framework for a specific web application. In our case, the exported rules are read by the server-side monitor. An example of a result file can be found in section 6.5.5.

SNPmonitor. The *SNPmonitor* is our generic monitor module approach which provides RBAC with SNPs for web applications considering modeled access control semantics. Basically, this module is responsible to decide whether or not a user is allowed to get access to a protected resource. The decision making is based on the web user's session information (e.g., previously visited location, assigned user roles etc.) and a policy file (e.g., generated by MagicSNP). In order to ensure robustness, our monitor module also handles any kind of access constraint violation: the web user is redirected to a corresponding error page including an appropriate error message with possibility to go back to its previously visited navigation context.

Technically, our SNPmonitor is implemented as a Java EE application, using the Spring Framework [91]. The code of a client application which should be safeguarded does not have to be touched; the monitor just has to be added as a filter to the Java EE deployment descriptor. Using a URL-pattern, it is also possible to shield a certain part of a web application, e.g., web pages stored in a `protected/*` directory.

SNPpolicyTester. In order to test already defined SNP policies for a specific web application, as mentioned in section 6.5.3, we developed a testing tool called *SNPpolicyTester*. It parses a given security policy file and searches for false positive and false negative access control behavior. Therefore, it navigates through every available SNP with every defined user role trying to leave the SNP on each navigational node by requesting every possible illegal node in this context. As a result, we get a detailed log file which allows a quick identification of traces that are possible although they should be prohibited.

Additionally, we analyzed the runtime performance of our testing tool using a benchmark client based on the TPC-W Benchmark[216]. Consequently, we are able to compare the result with the complexity of our test generation process according to section 6.5.3: figure 6.12 depicts the average result of benchmarks we performed with two user roles regarding ten, twenty, thirty and forty navigational nodes. The result corresponds to the expected complexity of $\mathcal{O}(rn^3)$.

6.5.5 Case Study

Smart grids use information and communication technology to optimize the transmission and distribution of electricity from suppliers to consumers, allowing smart generation and bidirectional

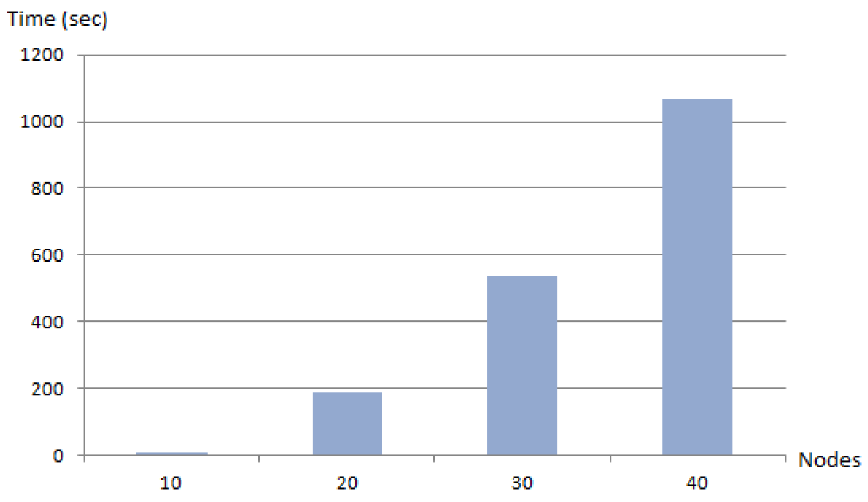


Figure 6.12: SNPs: average benchmark result of SNPpolicyTester

power flows – depending on where generation takes place. Thus, the smart grid enables financial, informational, and electrical transactions among consumers, grid assets, and other authorized users [141]. The smart grid integrates all actors of the energy market, including the customers, into a system which supports, for instance, smart consumption in cars and the transformation of incoming power in buildings into heat, light, warm water or electricity with minimal human intervention. Smart grids represent a potentially huge market for the electronics industry [184]. Two basic reasons why the attack surface is increasing with the new technologies are: (a) The smart grid will increase the amount of private sensitive customer data available to the utility and third-party partners and (b) Introducing new data interfaces to the grid through meters, collectors, and other smart devices create new entry points for attackers. For a more detailed discussion on security issues arising in this context see [62]. See also [104] for a current version of proposed technologies to solve this power systems management and associated information exchange issues. In the following, we model a scenario in this domain, the SmartGrid Bonus Application.

Basically, our SmartGrid Bonus Application represents a prototype of an energy offer management including optional bonus handling, as already described in section 5.1.

In order to model the data structure managed by our case study, we use UWE's Content model. Basically, it comprises two domain classes, `EnergyOffer` and `BonusProgram`, which are also used in figure 6.13. An instance of the class `EnergyOffer` represents a specific energy offer launched by an energy provider including start and end date. Each object of `EnergyOffer` can include an arbitrary number of `BonusProgram` instances. A `BonusProgram` instance stands for an additional bonus customers get, after buying the corresponding `EnergyOffer`.

In order to model RBAC constraints we use UWE's Basic Rights model, depicted in figure 6.13. Basically, it uses classes of the Content model on the left-hand side in combination with user roles on the right-hand side. Access permissions were defined by stereotyped dependencies: for our application, a provider has no restricting constraints. By contrast, there is only a limited set of permissions for users taking on the role of a customer: they are only allowed to read instances of the class `EnergyOffer` and to call the methods `buyOffer()` and `generateBonusCode()`. These permissions represent the basis for a customer to list all available energy offers, to buy a

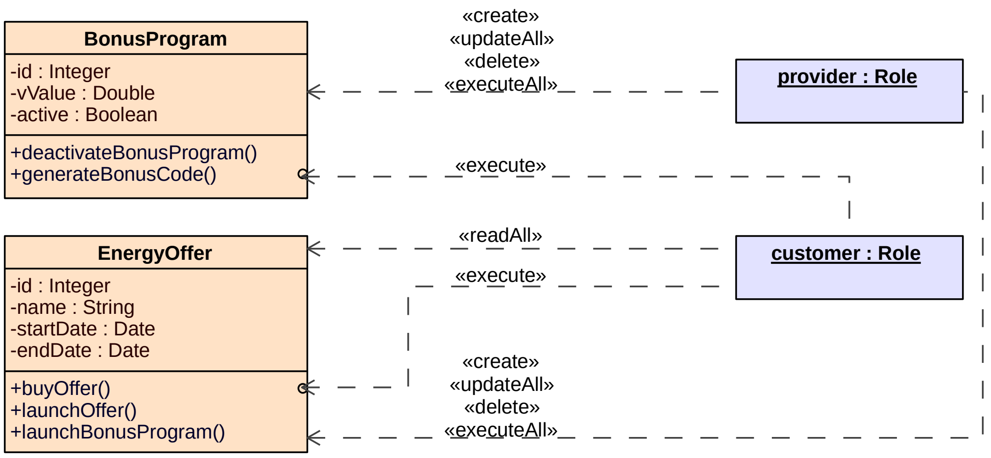


Figure 6.13: SNPs: SmartGrid Bonus – Basic Rights model

specific offer and to eventually get a bonus code. In order to define constraints like "a customer can only get access to a bonus code after he bought an energy package" we now have to define navigational access control policies using SNPs.

Therefore, we use UWE's Navigation model as described in section 6.5.2. For our web application, the navigation structure should start at a login page. After completing the authentication successfully, customers should be redirected to an internal page where they can have a look at a list of offers. If they decide to accept an offer they have to give their consent, before a confirmation is shown. In case the energy offer was connected to a bonus program, a page containing the bonus code is displayed before the final confirmation.

Notice that for our case study we make the assumption that names of pages correspond to names of states and we do not model parameters – our monitor then just forwards given parameters, if any.

Figure 6.14 depicts our Navigation model. The outermost state stands for the whole application and is called `SmartGridBonusApplication`. Navigational nodes, represented by the states on the innermost level, are grouped by three main areas or parent states: `LoginArea`, `ProviderArea` and `CustomerArea`.

Every web user can access the login context node `loginViaPasswordForm` which is inside the `LoginArea` indicated by the {isHome} tag. Inner states are tagged by {unauthorizedAccess= Error} which represents the default violation node. Logged in users with the role `customer` can access the whole `CustomerArea` indicated by the inherited {roles} tag. In addition, they must follow the SNPs as defined by the transitions between the navigational nodes to be allowed to request a protected node. This means, e.g., to get access to `showBonusCode` the user has to be associated to the role `customer` and he must have been on `buyEnergy` right before. In order to get access to `customerHome` the user needs to have the same role but must have been on one of the nodes `loginViaPasswordform`, `showEnergyOffers` or `showConfirmation` right before and so on. Otherwise, the user gets redirected to the `error` state as defined in the tag {unauthorizedAccess}. Each user which enters the constraint violation node `error` gets logged out automatically as indicated by the entry event `entry / logout()`.

SNPs for providers are modeled in an analogous manner as depicted in the lower-left corner of figure 6.14.

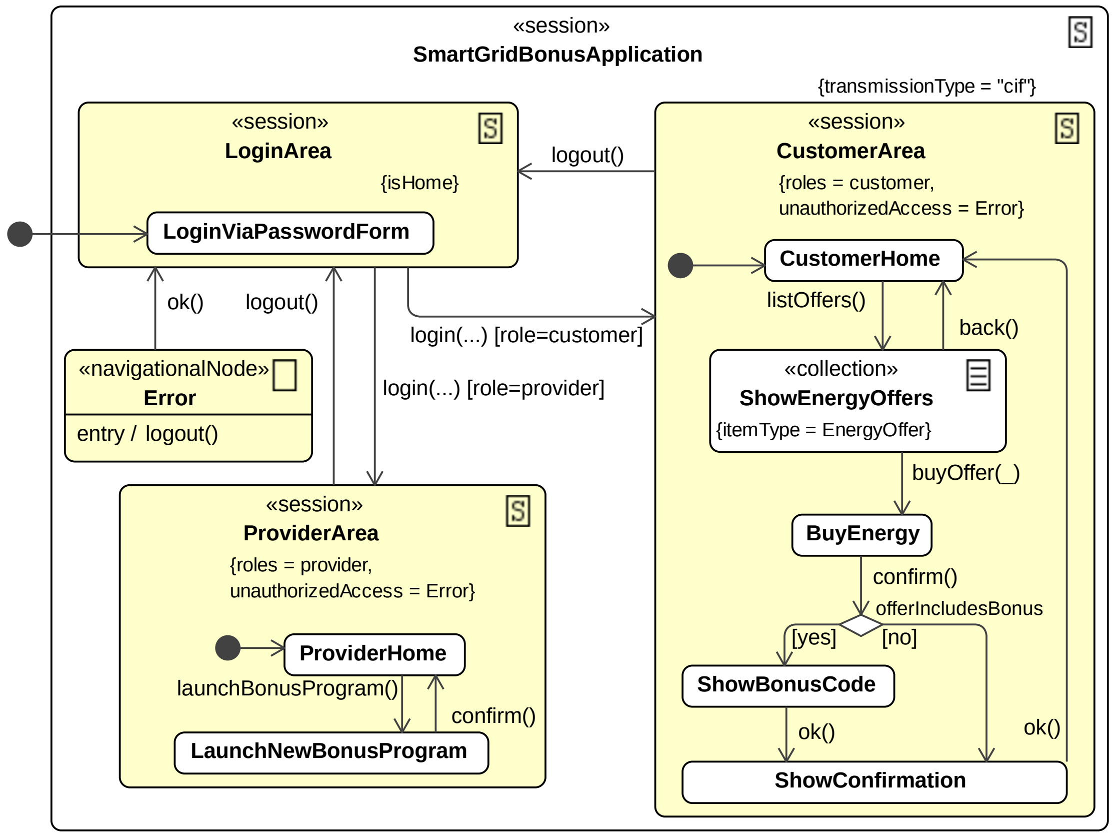


Figure 6.14: SNPs: SmartGrid Bonus – Navigation model

The following listing shows an excerpt from the navigation rule file which is generated by our tool *MagicSNP* from the state machine shown in figure 6.14.

```
navigation.file={
  "_comment":"Build time: 10.10.2012 11:12:38",
  "application":"SmartGridBonusApplication",
  "locations":[...
    {"location":"showEnergyOffers","violation":"error","home":false,
     "rules":[{"role":"customer",
              "pre_visited":["customerHome"]}]},
    {"location":"buyEnergy","violation":"error","home":false,
     "rules":[{"role":"customer",
              "pre_visited":["showEnergyOffers"]}]},
    {"location":"showBonusCode","violation":"error","home":false,
     "rules":[{"role":"customer",
              "pre_visited":["buyEnergy"]}]},
    {"location":"customerHome","violation":"error","home":false,
     "rules":[{"role":"customer",
              "pre_visited":["showEnergyOffers",
                           "showConfirmation",
                           "loginViaPasswordForm"]}]},
    {"location":"showConfirmation","violation":"error","home":false,
     "rules":[{"role":"customer",
              "pre_visited":["buyEnergy",
                           "showBonusCode"]}]},
    ...],
  "default_violation":"error"}
```

The rule file contains information for a generic monitor to provide navigational access control including SNPs for a web application: The default violation node is `error`, defined by the attribute `default_violation` on the outermost level. Furthermore, every single location entry holds the corresponding violation node and a list of access rules. Each rule entry represents a user role that is allowed to access the current navigational node. In addition, the attribute `pre_visited` specifies navigational nodes the user is allowed to come from. The rule file can be imported in our *SNPmonitor* as well as in our *SNPpolicyTester*.

6.6 Summary and Related Work

We answered the second part of RQ3, namely "How can resulting models be used in the development process?". Security models can serve as a basis for manual implementation of applications, as well as for a model-driven process. In this chapter, we showcased the generation of various artifacts based on UWE security models.

Artifact Generation. First, we experimented with a textual notation for UWE called TextualUWE that is represented as an internal DSL in Scala. It was created with the thought in mind that UML with its inaccuracies might not be the best choice for modeling security. However, it turned out that at the high level of abstraction that UWE models are located at, possible model inconsistencies due to the nature of UML do not play a major role. An example of a DSL in the area of web applications is WebDSL [93], a DSL from which Java web applications can be generated. It implements access control, but does not focus on the configuration of other security mechanisms. Second, we introduced the UWE2FACPL toolchain, which consists of an export of access control rules from the UWE Basic Rights model to policies written in XACML [149] and a transformation from XACML policies to more formal FACPL policies [130]. Third, we created the ACT toolchain. For exported XACML policies, requests were generated and tested using a Policy Decision Point (PDP). Thereby, the final step of the toolchain is able to detect inconsistencies. We integrated the ACT toolchain as well as the UWE2FACPL toolchain in a tool workbench, called SDE [193]. Many other approaches exist that address the complexity of XACML, as e.g., graphical editors like a Scratch-based graphical policy editor [142] or the UMU-XACML-Editor[21]. The advantage of starting our toolchains at UWE's Basic Rights model is that it abstracts from the complex, tree-like structure of XACML policies and not only from writing XML, like these editors do. Fourth, we sketched a transformation called ActionUWE that transforms UWE models to ActionGUI models [11]. However, we abandoned our original plan to implement a transformation from UWE to ActionGUI in order to generate code from ActionGUI models, as it turned out that it would not be efficient, when compared with traditional coding of the limited kind of applications that could be generated. Fifth, we modeled, tested and enforced navigation paths, i.e., the sequence of navigational nodes a web user visits within a web application. The closest related work is published by Braun et al. [31], who created a robust approach for SNPs for MVC-based web applications where policies are specified using a textual notation instead of UWE's Navigation model.

Most transformations that are described in this chapter have already been published as joint work. Special thanks go to all coauthors of these publications.

Outlook. As our tools to generate artifacts serve as proofs-of-concepts, there is room for improvement. In addition, future work could examine, how tools can assist in round-trip engineering, i.e. in keeping UWE models up-to-date in case associated code implies that design decisions might have been changed. It also would be interesting to extend MagicUWE in a way that it derives security features from tools that manage security goals, like Adamant[22]. These features could then be suggested to the modeler for refining or adding them to an application's UWE model.

[21]UMU-XACML-Editor. https://sourceforge.net/p/umu-xacmleditor/

[22]Adamant. http://adamant.q-e.at/

Part IV

Conclusion

Chapter 7

Summary

The approach presented in this thesis consists of three parts, as depicted in figure 7.1: SecEval, a security-aware conceptual evaluation framework; SecWAO, an ontology for the domain of web engineering, which instantiates SecEval, and a security extension for the modeling language UWE that references concepts of SecWAO in order to document design decisions for secure web applications. In the following, we briefly summarize the answers to the research questions posed in the introduction (section 1.2).

RQ1 Which key concepts emerge while developing software? Which properties do these concepts have and how are they related to each other? How can this knowledge be applied to evaluate concepts?

We identified the key concepts for developing software as assets, methods, tools, notations, security properties, vulnerabilities and threats (so-called knowledge objects). Our Security Context model represents knowledge objects with their properties and relations. For example, a method can be supported by a tool to detect a vulnerability that endangers a security property, which was intended to be a property of an asset (cf. figures 3.4ff.). Properties of methods and tools can best be described with attributes according to the phases of the Software Development Life Cycle (SDLC) they are used in, e.g., for a design method it is worth knowing if it is able to generate artifacts. The Security Context model is part of our novel conceptual framework, called SecEval.

In addition to the Security Context model, SecEval supports the process of evaluating concepts, as e.g., selecting appropriate methods or tools within the development process. Therefore, the Data Collection model can be used to plan and document research questions and queries on used resources, such as search expressions for search engines. The Data Analysis model involves the analysis strategy that is used, which includes filtering and categorizing data. As the Data Analysis model is connected to the Security Context model, data can easily be reused from previous evaluations or compared with them.

We demonstrated that the Security Context model can also smoothly be used as an underlying structure for an ontology or a knowledge base, which was implemented as a wiki-like web application. Furthermore, SecEval models can easily be extended to meet needs of additional domains, as shown for Moody's method evaluation approach and OWASP's Risk Rating Methodology.

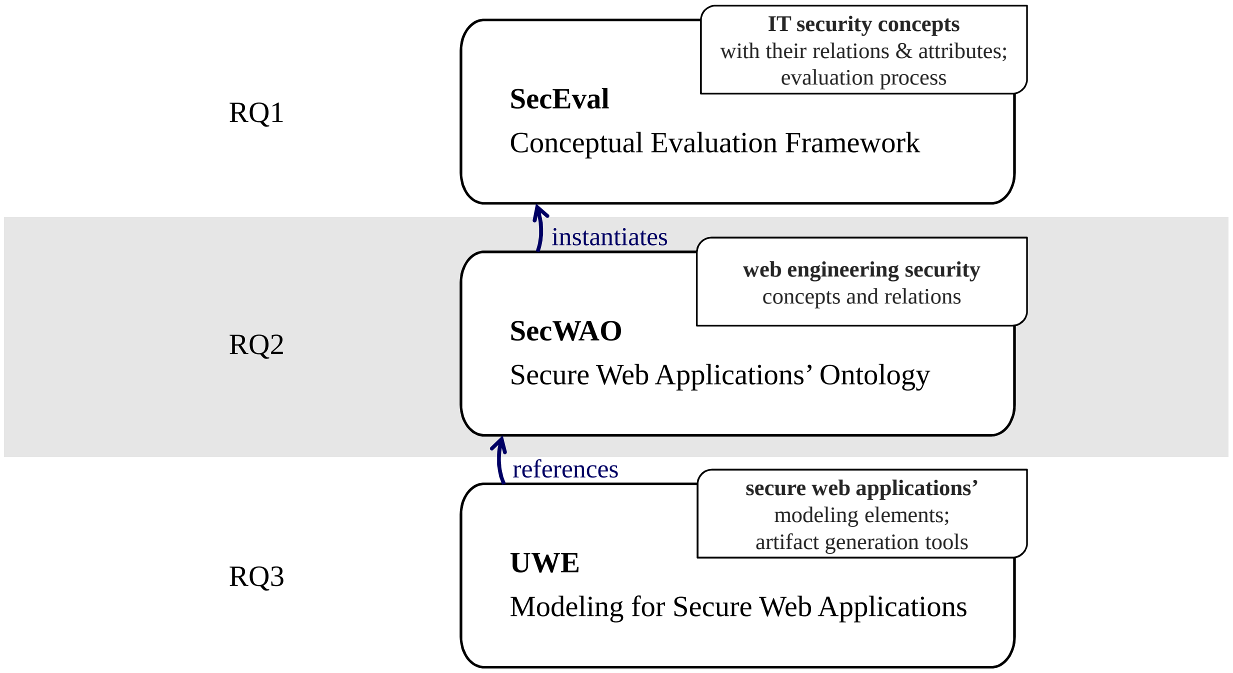


Figure 7.1: The presented approach, including SecEval, SecWAO and our UWE extension

RQ2 How are assets, security engineering methods, notations, tools, security properties, vulnerabilities and threats related in the domain of web application security?

Even without any extensions, SecEval's Security Context model turned out to be expressive and yet concise enough to serve as an appropriate TBox for our Secure Web Applications' Ontology (SecWAO). Consequently, we present concrete relations in the ontology, e.g., the security property "control flow integrity" can be endangered by Cross-Site Scripting (XSS), a vulnerability that may be exploited by a common threat like JavaScript code that steals assets such as user data displayed on a website (cf. figures 4.1ff.). By presenting a comprehensive view of web engineering security, SecWAO can give thought-provoking impulses for security-related decisions that should be taken during the development process of web applications.

RQ3 How can security aspects of web applications be expressed? How can resulting models be used in the development process?

Security of web applications can be expressed using distinct models that provide different views on the architecture of a web application, including its security aspects. We thus extended the existing modeling technique UWE that already provided different views on web applications. Beyond others, UWE defines the Requirements model for modeling use cases, the Content model for the application's data structure and the Navigation model for defining the paths that a web user can navigate on through the web application. We systematically explored SecWAO's security properties and methods in order to model them as UML stereotypes and tags in UWE. This creates the possibility for web engineers to document security requirements, abstract design decisions as well as concrete plans regarding employed methods or tools. For example, the methods or tools that are used or should be used to prevent XSS vulnerabilities within a certain web application's component can be documented using the {xssPrevention} tag. This is possible due to the UWE profile where we added this tag to UWE's stereotype ≪component≫ that inherits from UML components – a stereotype that is supposed to be used in UWE's Content model.

The resulting models can be used in the development process in various ways, as our case studies in the areas of smart grid, cloud and patient monitoring confirmed. Besides of modeling, several artifacts can be generated from UWE models, as e.g., access control policies can be generated from UWE models, such as XACML or FACPL policies. In addition, tools can export artifacts for testing and enforcing secure navigation paths, i.e., the sequences a web user is restricted to while using a web application.

In the end, we think that the basis for developing complex and yet sufficiently secure software is not only the acquisition of professional knowledge about IT security, but also the awareness of dependencies and the willingness to consciously make, document and implement design decisions. Our approach supports diligence in secure software development, as it bridges the gap between vague security knowledge and the conscious protection of web applications. Therefore, all methods and notations we provide are designed to be usable as a whole or in parts in any software development process, whenever practitioners want to take advantage of them.

Chapter 8

Future Work

In this chapter, we like to point out starting points for future research and indicate open questions. On the one hand efforts should be made to adapt our approach to future changes and to extended tool support. On the other hand, securing software is not a purely technical endeavor and it would help to better understand long-term human interactions with development methods that try to raise security awareness as well as social expectations regarding the protection of data and the accepted dependence on technology in favor of technical progress.

Continuous Adaptation and Further Development. Security of web applications is a dynamic field; every year new threats, new vulnerabilities and new security methods are emerging. In order to remain useful, SecWAO should encompass arising concepts. As a consequence, it might be desirable to further extend UWE to cope with these concepts. Consequently, it could also be interesting to compare different versions of SecWAO in the distant future to analyze what has been considered central for the domain of web application security over time.

Beyond mere adaptation, further development of SecEval and SecWAO might be used for structuring and illustrating the contents of a kind of "Guide to the Software Engineering Body of Knowledge (SWEBOK)" [29] for secure software engineering. Besides, it would be interesting to bring together some concepts of SecEval and SecWAO and existing ontologies, structures and standards from, e.g., threat databases, weakness databases, vulnerability databases, threat intelligence approaches, safety (IEC 61508[1]) or general quality factors (cf. ISO/IEC 25010[2]). In addition, the use of UWE models in quickly evolving environments, as e.g., in agile projects, could be examined. Breu's work [32] and other research that focuses on change propagation between modeling views, between model and code and within code (i.e., refactoring approaches) could be a source of inspiration for this task.

Extended Tool Support. In general, UML tooling could be improved to a great degree, as even working with commercial tools like MagicDraw [147] is often time-consuming and annoying, because it is not possible to use layers for hiding momentarily irrelevant details and the styling options are very limited. This experience is consistent with findings from Mohagheghi et al. [135] and Chaudron et al. [53]. The latter conclude that the "decision whether or not to modeling in a

[1]IEC 61508. http://www.iec.ch/functionalsafety/

[2]ISO/IEC 25010. http://www.iso.org/iso/catalogue_detail.htm?csnumber=35733

software project is not so much driven by cost factors. Instead, this decision is strongly influenced by a hygiene factor: inadequate tooling.”. Ideas for future work regarding modeling and modeling tools are elaborated by Mussbacher et al. [139].

Regarding the further development of UWE, a closer connection to code is desirable, not only for round-trip engineering, but also for drawing the developers’ attention to general decisions that were made, without the need to leave the IDE, as also discussed by France et al. in [86]. We think it could be worthwhile to investigate in warnings that appear in the IDE on the basis of the models. Another approach would be to provide means for marking parts of code or configuration files that are key for realizing a certain security property and should only be changed with increased attention, e.g., requiring the 4-eyes principle.

For working with SecEval, we suggest enhancing our wiki-like knowledge base. Regarding the maintenance of SecWAO, collaborative editing and simple accessibility is crucial, which could be achieved by porting the ontology to a web-based system. A tagging system that allows to tag (or briefly comment) every bit of information on the spot including the possibility for subsequent visitors to vote tags / comments up or down could help to find out if new relations should be introduced (cf. [225, ch. 4.1]). It might be worth a try to extend semantic editors, such as WebProtégé [206], in a way that elements of our ontology can be browsed in a 3D tag cloud that can be filtered and rotated. For each element, links to existing information (from OWASP, Wikipedia, Cybrary[3], . . .) could be made available and it should be possible to create high-quality snapshots that can be used as infografics or for discussions with peers.

Long-term Evaluation. From our practical experience in using our approach, a well-informed choice is easier to be made if it is feasible to get an overview of existing security engineering methods, notations and tools as well as security properties, vulnerabilities and threats. Furthermore, the web applications’ level of security is assumed to benefit from having a comprehensive view of decisions that were made, as long as they are connected to relevant parts of the application’s architecture. However, long-term evaluation is needed to show that the effect we observed in our case studies can be repeated in large projects that run over many years.

It would also be interesting to conduct empirical studies in order to find further ways to support conscious, secure software development, i.e. methods to make practitioners aware of important facts, and decisions that should be taken. Thereby, the challenge is to develop techniques that are able to rate risks in a certain context so that practitioners are guided to pay attention to high-priority tasks first.

Examining Connections to Other Disciplines. As it is not possible to secure all systems and applications a hundred percent and to restrict all functionality in a way that it can only be used for legal purposes, collaboration with social sciences is essential. But then, legal and political requirements influence the work of software and security engineers as well.

In the areas of politics and law, two contrary social expectations collide: On one side, protection of data and with that privacy, and on the other fair jurisdiction and prosecution of criminals who intercepted critical data, which usually requires traceability. Article 12 of the Human Rights Declaration[4] states that “No one shall be subjected to arbitrary interference with his privacy, family, home or correspondence, nor to attacks upon his honor and reputation. Everyone has

[3]Free IT and cyber security learning. http://cybrary.it/
[4]The Universal Declaration of Human Rights. http://www.un.org/en/documents/udhr/

the right to the protection of the law against such interference or attacks." In the realm of web security, conflicting examples can easily be found, as crimes can be committed using the Internet, where the actor's actions can be concealed using the same technical means as those that ensure the privacy of innocents. Thus, we think that this field needs careful consideration of technical feasibility and its practical implications to counteract what Will et al., the authors of the book "Blackhatonomics", describe as "a great time to be a cybercriminal: Not only have the laws of most countries not yet caught up with the technology (let alone the crime), but the politics of creating cybercrime laws are mired in a power struggle between agencies in single countries, and are struck in an absolute gridlock when more than one country is involved" [92, p.2].

Another huge research topic considers social expectations regarding technical progress and the thereby increasing dependence on technology (especially on cyber-physical systems) that cannot yet be secured adequately. Daily news continuously report on recent, severe security incidents as data theft on web applications or successful attacks on critical infrastructure like cars [134], central heatings[5] or cardiac pacemakers[6].

Many questions arise from this situation, as e.g., the following: Is it wise to do everything that is technically possible? To what extend does our society profit, due to cost reduction and additional features, from interconnected, "smart" devices that can be controlled via web applications or mobile apps, despite high security and hence safety risks? If everything gets connected and controlling technological systems means exercising power, how does this interfere with the principles of democracy? How big is the willingness of users to refrain from using certain features, because they cannot be appropriately secured? When do people start to despair or stop to speak out freely, because they feel helpless regarding the fact they might be spied upon without noticing? Could the overall situation be improved by intensifying product liability, including legal responsibility for data breaches in commercial software[7]? For software and security engineers, all these questions are relevant for taking an unequivocal stand on software features they are inclined work on, which could mean to speak up, in case professional ethics are violated, e.g., by a technical security risk that seems to be irresponsibly high. In any case, we think it is of supreme importance to examine how a basic understanding and awareness of web security can be brought not only to developers, but also to users so that they can take informed decisions.

We expect IT security to remain an important topic, in particular as humans tend to underestimate risks (optimism bias), due to the feeling that outcomes can be controlled, even if

[5]Vaillant-Heizungen mit Sicherheits-Leck. (German) http://www.heise.de/-1840919.html

[6]Hacking Pacemakers. http://spectrum.ieee.org/podcast/biomedical/devices/hacking-pacemakers

[7]"Information insecurity is costing us billions. We pay for it in theft: information theft, financial theft. We pay for it in productivity loss, both when networks stop working and in the dozens of minor security inconveniences we all have to endure. We pay for it when we have to buy security products and services to reduce those other two losses. We pay for security, year after year.
The problem is that all the money we spend isn't fixing the problem. We're paying, but we still end up with insecurities. The problem is insecure software. It's bad design, poorly implemented features, inadequate testing and security vulnerabilities from software bugs. The money we spend on security is to deal with the effects of insecure software. And that's the problem. We're not paying to improve the security of the underlying software. We're paying to deal with the problem rather than to fix it. The only way to fix this problem is for vendors to fix their software, and they won't do it until it's in their financial best interests to do so. Today, the costs of insecure software aren't borne by the vendors that produce the software. In economics, this is known as an externality, the cost of a decision that's borne by people other than those making the decision" [186, ch. 1, p. 12].

sufficient influence through controls is unrealistic [131]. However, approaches like those presented in this thesis provide support for taking security into consideration from the beginning of the development process, aiming at the prevention of vulnerabilities in the first place.

Part V

Appendix

Appendix A

Attribute description of SecEval's Security Context Model: Methods and Tools

In section 3.2.1, SecEval's Security Context model is introduced including most class' attributes. However, we just briefly mentioned some attributes that describe methods or tools according to the phases in the SDLC they belong to. In the following two sections, these attributes are described successively.

A.1 Methods

In the following, attributes shown in figure 3.5 (`MAreasofDev`) are detailed:

RequirementsM.comprisesElicitation is set to true if a method is used for requirement elicitation.

RequirementsM.comprisesAnalysis is set to true if a method comprises the analysis of requirements.

RequirementsM.comprisesSpecification is set to true if a requirements specification is part of a method.

RequirementsM.comprisesManagement is set to true if the management of requirements is defined by a method.

RequirementsM.levelOfDetail describes how detailed the method's requirements features are. As this is a subjective value, we recommend adding a short explanation.

DesignM.isUsedForCommunication is set to true if a design method is also used for communication purposes, as e.g., UML.

DesignM.canGenerateArtifacts is set to true if the method specifies the generation of artifacts (code, configuration files, . . .) out of the design of an application.

ImplementationM.supportedLanguageTypes types of languages supported by a method. An example would be: "functional programming languages" are supported by a method that auto-completes fragments of code.

ImplementationM.inspectsArtifacts is true if the method relies on read access to artifacts, as e.g., program code.

ImplementationM.writesArtifacts is true if the method writes artifacts, as in the above-mentioned example of code completion.

ImplementationM.changesArtifacts is true if the methods changes existing artifacts.

ImplementationM.completesArtifacts is true if the method completes already existing artifacts. An example is the auto completion used in development environments.

ImplementationM.givesSecurityAdvice is true if security-related hints are provided during the implementation.

TestingM.isTestingWebApp is true if a method describes how to test web applications.

TestingM.isTestingNetwork (same with network-related testing)

TestingM.isTestingSystem (same with system-related testing; the system refers to a soft- or hardware system)

TestingM.isWhiteboxTest is true if the method is based on white-box tests, which means that the tester has full access to all internal details of a system.

TestingM.isBlackboxTest is true if the method is based on black-box tests, which means that code and other internal details cannot be accessed by a tester.

TestingM.isGreyboxTest is true if the method uses tests that are a mixture of white-box and black-box tests, which means that the tester can access some internal details of the system, but not all.

DeploymentM.versionTrackingApproach describes the approach that is used for tracking available versions of a piece of software.

DeploymentM.adaptionProcessToNewEnv describes the process that is specified for adapting software to a changed environment, e.g., if hardware is changed.

DeploymentM.updateProcess describes the process for updating software.

AssuranceM.usedFormalisms lists the formal methods that are used by an assurance-related method. This is a shortcut in case we do not want to model those formalisms as a method on its own.

AssuranceM.assuranceCriteria gives the criteria the assurance process relies on.

AssuranceM.canProvideCounterExample. In case formal methods are applied, it often makes sense to provide a counter example. The attribute is true in case the method can give at least one.

AssuranceM.canProvideProof is true if a formal proof for a certain criterion can be given.

ProtocolVerification.usedAdversaryModels. Protocol verification is an example how to go into more detail without changing the core of SecEval: e.g., the used adversary models can be specified with this attribute.

ProtocolVerification.usedEquation describes the equation that is used.

ProtocolVerification.usesFixedProperties is true if fixed properties are used for the protocol verification.

RiskAndCostM.usedIndicators names the indicators used to analyze security-related risks and costs.

RiskAndCostM.supportsRiskIdentification is true if the method gives advice how to identify risks.

RiskAndCostM.supportsRiskAssessment is true if the method supports risk assessment approaches.

RuntimeM.worksWithLanguages lists programming languages on which the method can be applied. Usually, methods that define techniques related to the runtime behavior of the targeted system are rather language-specific.

RuntimeM.canDamageSystem is true if the method describes how to damage a running system. This applies especially for attack-related methods.

RuntimeM.canInsertData is true if the method allows data to be inserted into the running system so that the system uses it.

RuntimeM.canAlterData is true if the method comprises changing data.

RuntimeM.canStealData is true if the method defines how to steal data.

RuntimeM.canInspectData is true if the method assumes that inspecting data is possible. This does not include that data can also be transferred outside of the system, which would be stealing.

RuntimeM.canInspectFlow is true if the flow of data can be inspected.

RuntimeM.canGenerateFlow is true if a data flow chart can be generated.

RuntimeM.canBlockFunctionCalls is true if the method specifies how to block function calls.

ServiceCompositionM.supporedTypesOfServices names services that are eligible to be composed.

ServiceCompositionM.securesServiceComposition is true if the method describes how to secure a service composition.

ServiceCompositionM.knowsAvailableServices is true if the concept is to know available services.

ServiceCompositionM.isCentralized is true if the method anticipates a centralized instance for being applied.

A.2 Tools

Attributes that describe methods also describe tools, due to the fact that tools (partially) support methods. This means, a tool is implicitly described by method's attributes, which leaves fewer attributes that are tool-specific. In the following, these attributes, which are depicted in figure 3.6 (`TAreasofDev`), are detailed:

RequirementsT.supportsHandwrittenSketches is true if handwritten sketches can be managed with the tool.

DesignT.generatedArtifacts lists artifacts (like code) that can be generated by the tool.

ImplementationT.worksWithLanguages names languages that are supported by the tool. Examples are tools that perform auto-completion on the code. The related method might be called "auto-completion on imperative languages" and a tool could support Java.

TestingT.targetLanguages lists languages the target system can be written in so that the testing-related tool can be applied.

TestingT.targetArchitectures describes how the system architecture has to look like in order to test it with the tool.

TestingT.targetFrameworks names target frameworks that are supported by the tool.

TestingT.usedExploitDatabases refers to databases containing up-to-date exploits.

DeploymentT.canInstallSoftware is true if tool is able to install software.

DeploymentT.canConfigureSoftware is true if tool is able to configure software.

DeploymentT.canAdaptSoftwareToChangedEnv is true if the tool can adapt software to a changed environment, e.g., a changed hardware environment or a changed server configuration.

DeploymentT.canUpdateSoftware is true if the tool can update software.

DeploymentT.canUninstallSoftware is true if the tool can uninstall software.

DeploymentT.hasVersionManagement is true if the tool can track versions of software, which means it manages versions of different software that can be used together.

DeploymentT.canCheckSecureInstallation is true if the tool can make sure that software was installed correctly so that security requirements are fulfilled.

DeploymentT.canCheckCorrectInstallation is true if the tool can make sure that software was installed correctly.

DeploymentT.isConnectedToConfigurationMngt is true if the tool is connected to or implements a configuration management system.

AssuranceT.canExportLaTeXproof is true if a proof is part of the assurance-related method and it can be exported as LaTeX formulae.

AssuranceT.canVisualizeAttacks is true if attacks can be visualized using graphical models.

RiskAndCostT.usedManagementInterfaces lists management interfaces used by the tool, e.g., an interface to an Enterprise Resource Planning (ERP) system.

RuntimeT.isInstalledOnTargetSystem is true if the tool is installed on the same system where the target software is installed which is accessed at runtime.

RuntimeT.canCoverOwnTraces is true if the tool is able to cover the own traces, as e.g., logfile entries or additional files that were downloaded. This usually applies to tools that are used for attacks.

RuntimeT.usedExploitDatabases lists databases containing known exploits.

ServiceCompositionT.supportsOfflineServices is true if the tool supports services that run offline, i.e., are not running at a server.

ServiceCompositionT.supportsOnlineServices is true if the tool supports services that run online, i.e., are running at a server (which is usually not the host where the service composition tool is installed).

Appendix B

SecEval Questionnaire

In the following, the questionnaire about SecEval's Security Context model is reprinted [42, appendix]. It was handed out at the NESSoS plenary meeting in Málaga (29.5.-31.5.2013). In the original version of the questionnaire, there was more space after each question so that the answers could be handwritten.

B.1 Security Engineering Method and Tool Evaluation

Our **aim** is to provide an approach for the evaluation of methods and tools for the engineering of secure software systems. In our approach we do not only distinguish methods and tools, but also notations. For an evaluation and comparison approach we need to define (1) the process of how to conduct a comparison and (2) the structure used to collect security-related data and metrics to analyze it. Therefore, we define a conceptual framework that comprises these three aspects: Security Context, Data Collection and Analysis. We depict the concepts and their relationships as a model (see Figure B.1 for an overview), so that we can instantiate concrete methods, tools and notations.

Additionally, we will compare other approaches with our framework in order to further adapt or extend it with the objective to make it more general. The **basic structure** of our model of the Security Context is as depicted in figure B.2.

The classes Method, Notation and Tool are depicted in the center. They inherit general attributes, as e.g., names and URLs, from the abstract class Mechanism (we are still looking for a better name to replace "mechanism", suggestions are welcome!). A tool can support methods and a notation can be used for several methods.

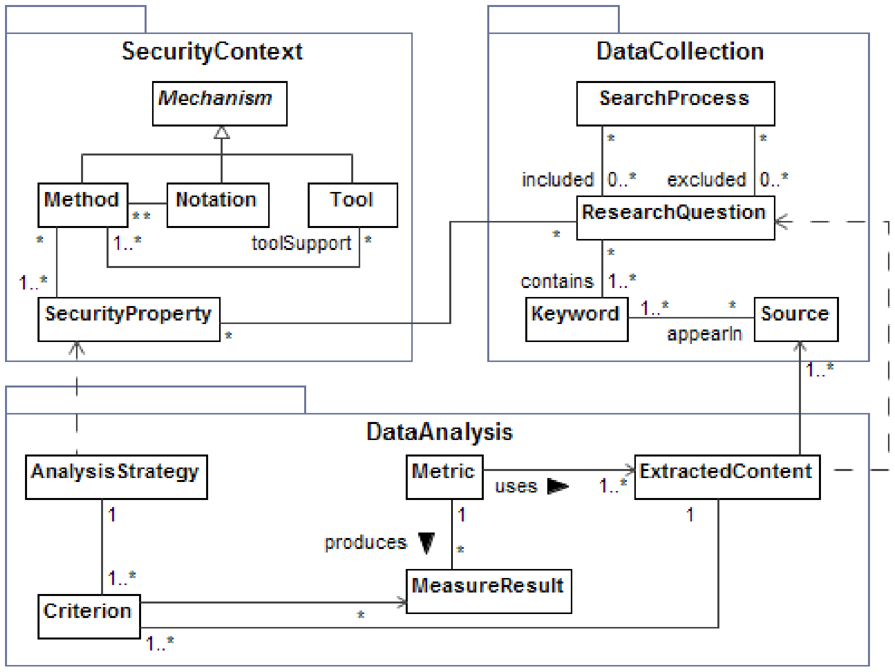


Figure B.1: Overview

Security features are shown on the left hand side of Figure B.2.

- A Security Property can be, e.g., authorization, authentication, integrity, etc. Several security properties can be enforced or attacked by a method.
- A Vulnerability can endanger security properties. Examples are XSS, SQL Injection, Buffer or Overflows, etc.
- A Threat can exploit vulnerabilities. Threats are kind of methods which are vicious.

In Figure B.3 resp. B.4, the classes **Tool and Method** are refined according to their usage in the Software Development Life Cycle (SDLC).

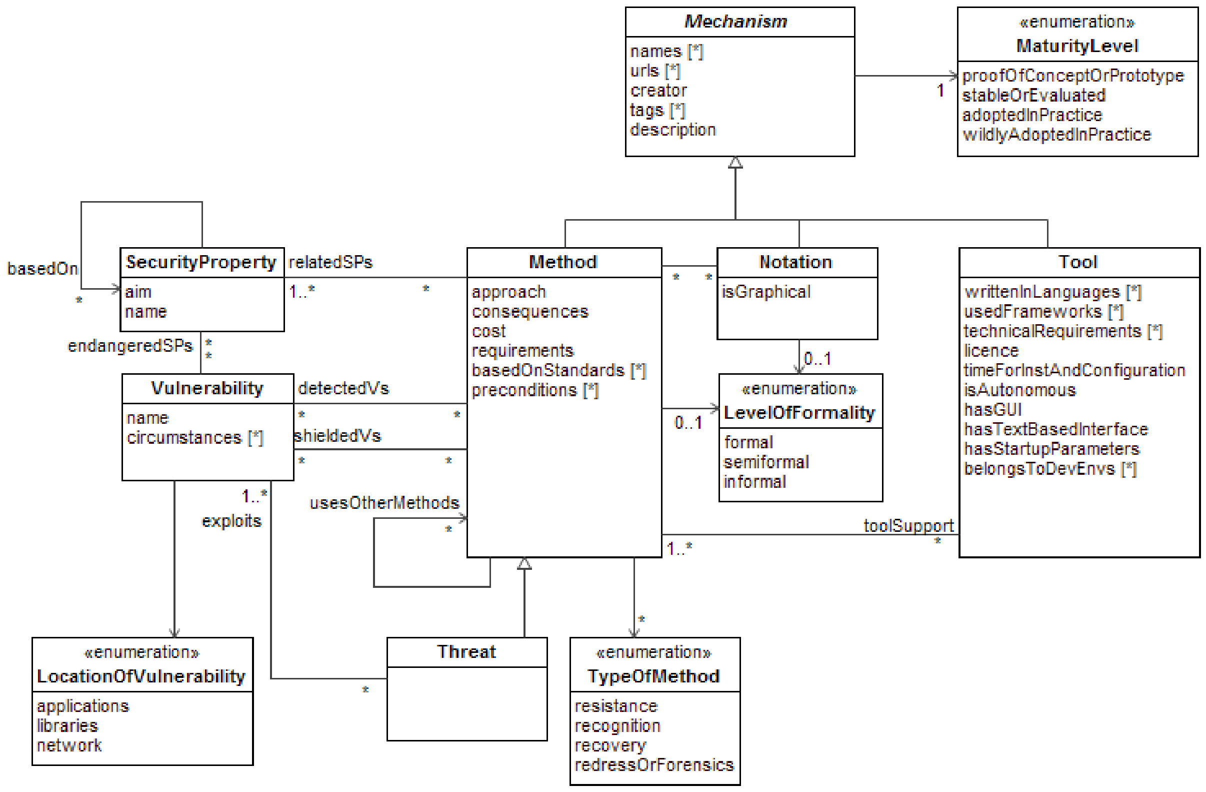


Figure B.2: Security Context

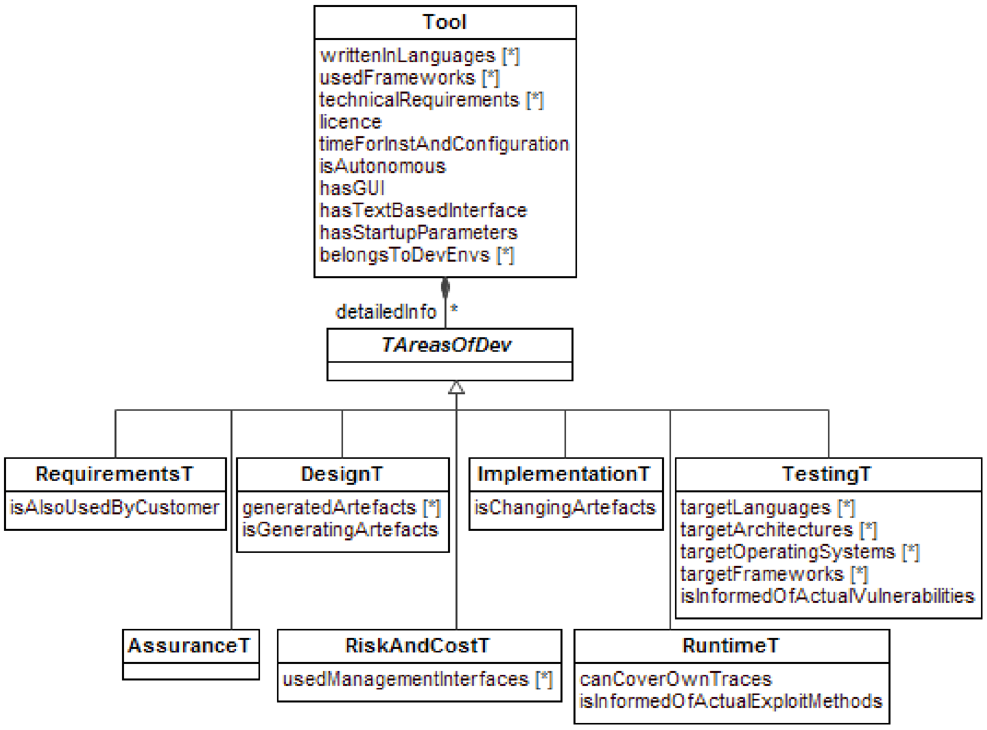


Figure B.3: Security Context: Details of Tools

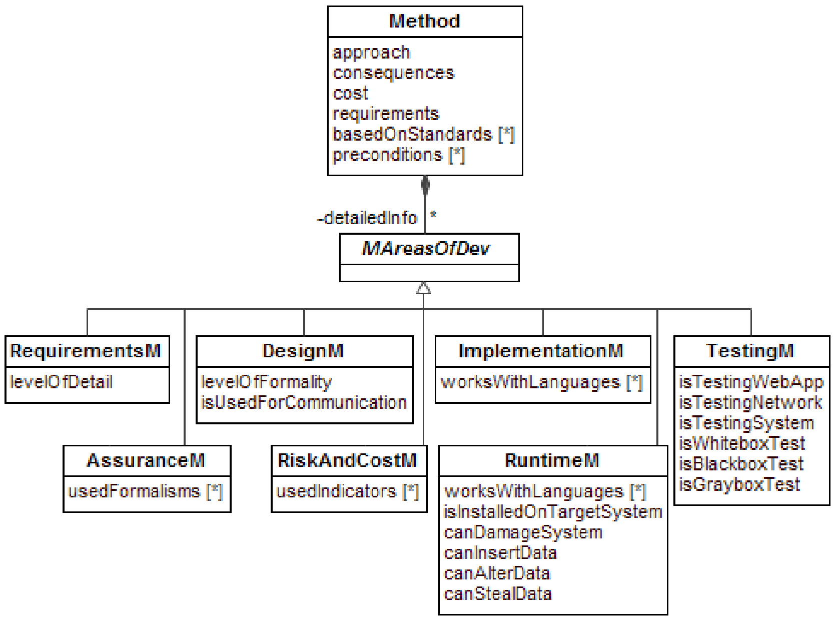


Figure B.4: Security Context: Details of Methods

B.2 Questions and Suggestions

1. Name:
   Partner:

2. Areas of security you are working in?

3. Can methods from your area be represented using our model?
   Provide examples of methods.

   Are concepts or relationships missing? Which?

4. Can tools from your area be represented using our model?
   Provide examples for tools.

   Are concepts or relationships missing? Which?

5. Can notations from your area be represented using our model?
   Provide examples of notations.

   Are concepts or relationships missing? Which?

7. Would you use our structure to evaluate tool, methods or notations in your area? If not, what would your approach look like?

   If yes, where do you see its strengths?

8. Can you suggest related work (esp. for managing tool and method portfolios for the area you are working in or general approaches to compare with our approach)?

9. General comments or improvements?

Appendix C

Excerpt of the UWE Profile

UWE models and their stereotypes and tags are defined in the UWE profile. This chapter depicts (excerpts) of diagrams that show the relation of stereotypes, tags and metaclasses in the UWE profile. The full profile can be downloaded from the UWE website [224], including the specification of the Presentation model and of the Process model, which are not in the focus of this thesis.

C.1 Requirements Model

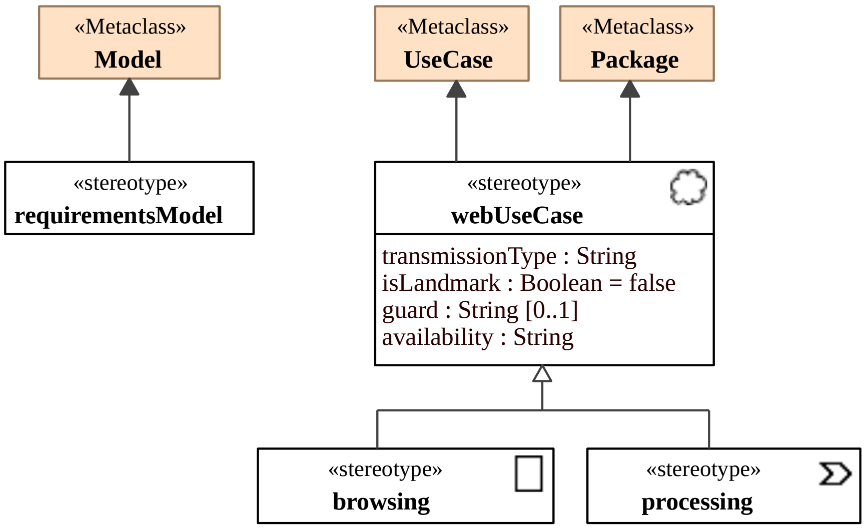


Figure C.1: UWE profile: Requirements model

C.2 Content Model

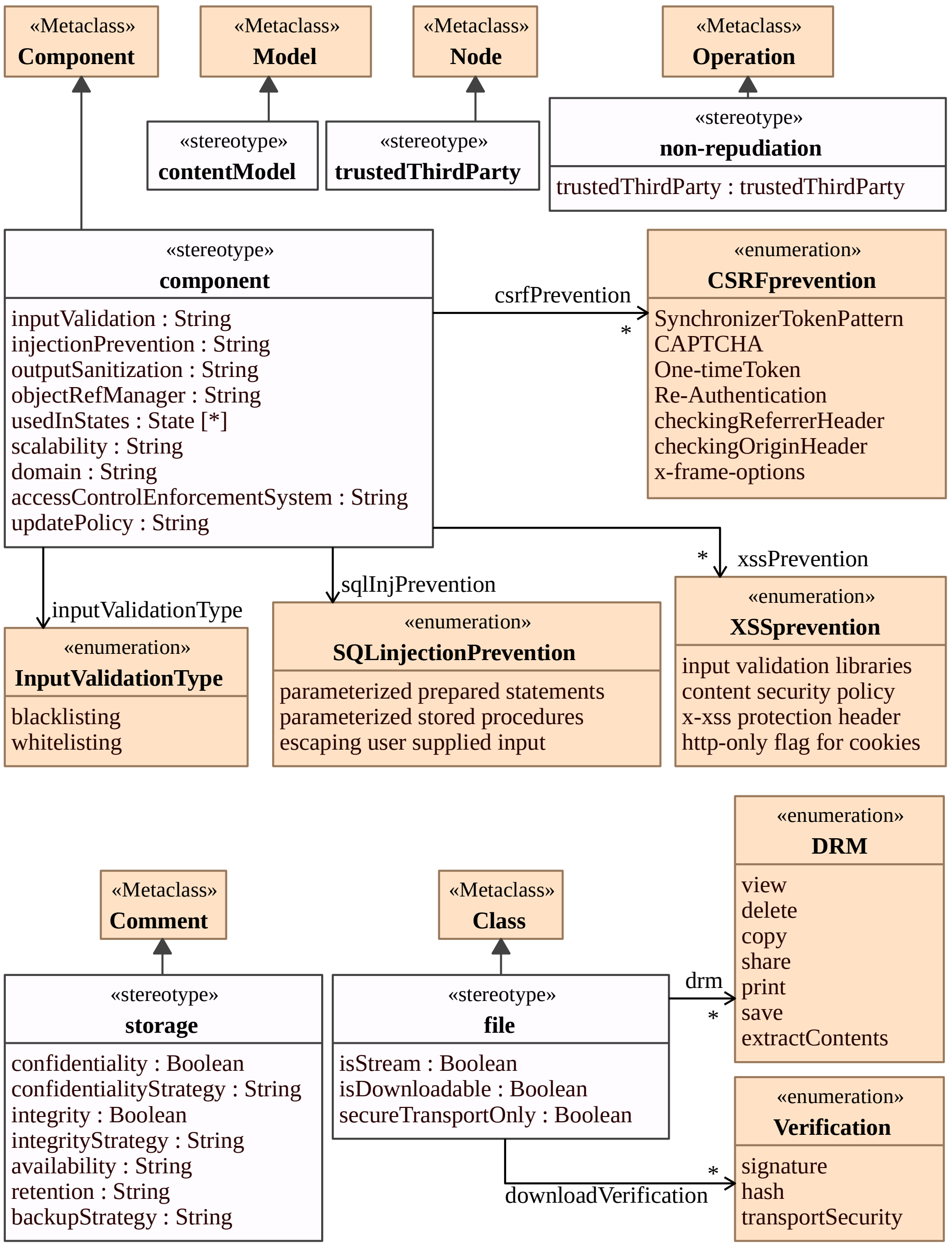


Figure C.2: UWE profile: Content model

C.3 User Model and Role Model

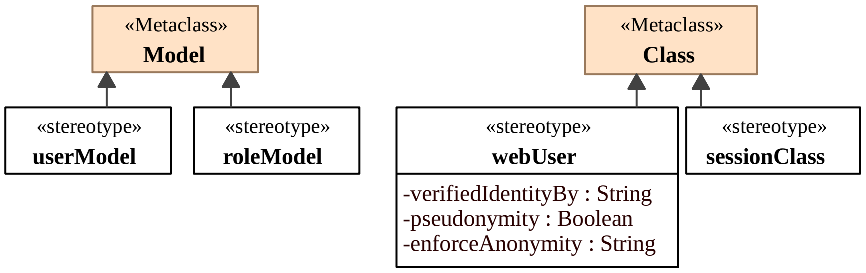


Figure C.3: UWE profile: User model and Role model

C.4 Basic Rights Model

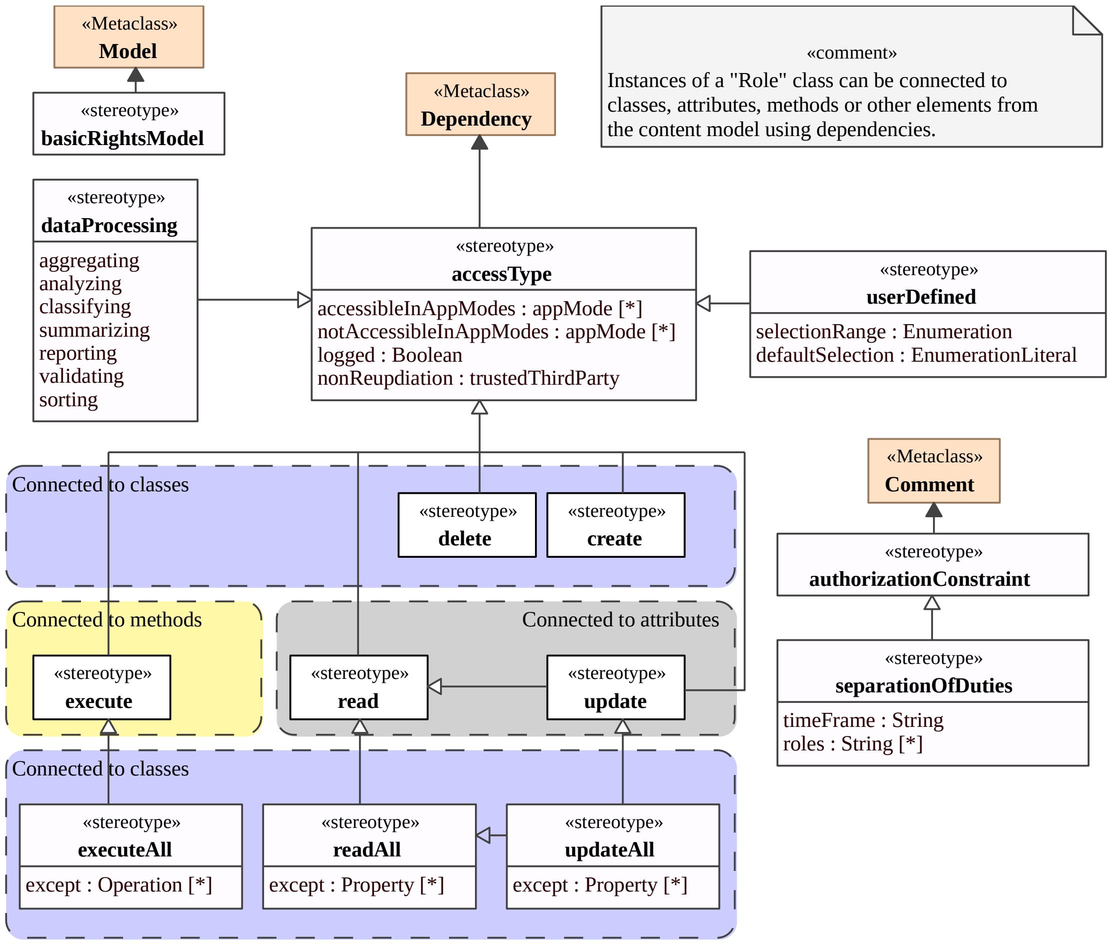


Figure C.4: UWE profile: Basic Rights model

C.5 Navigation Model

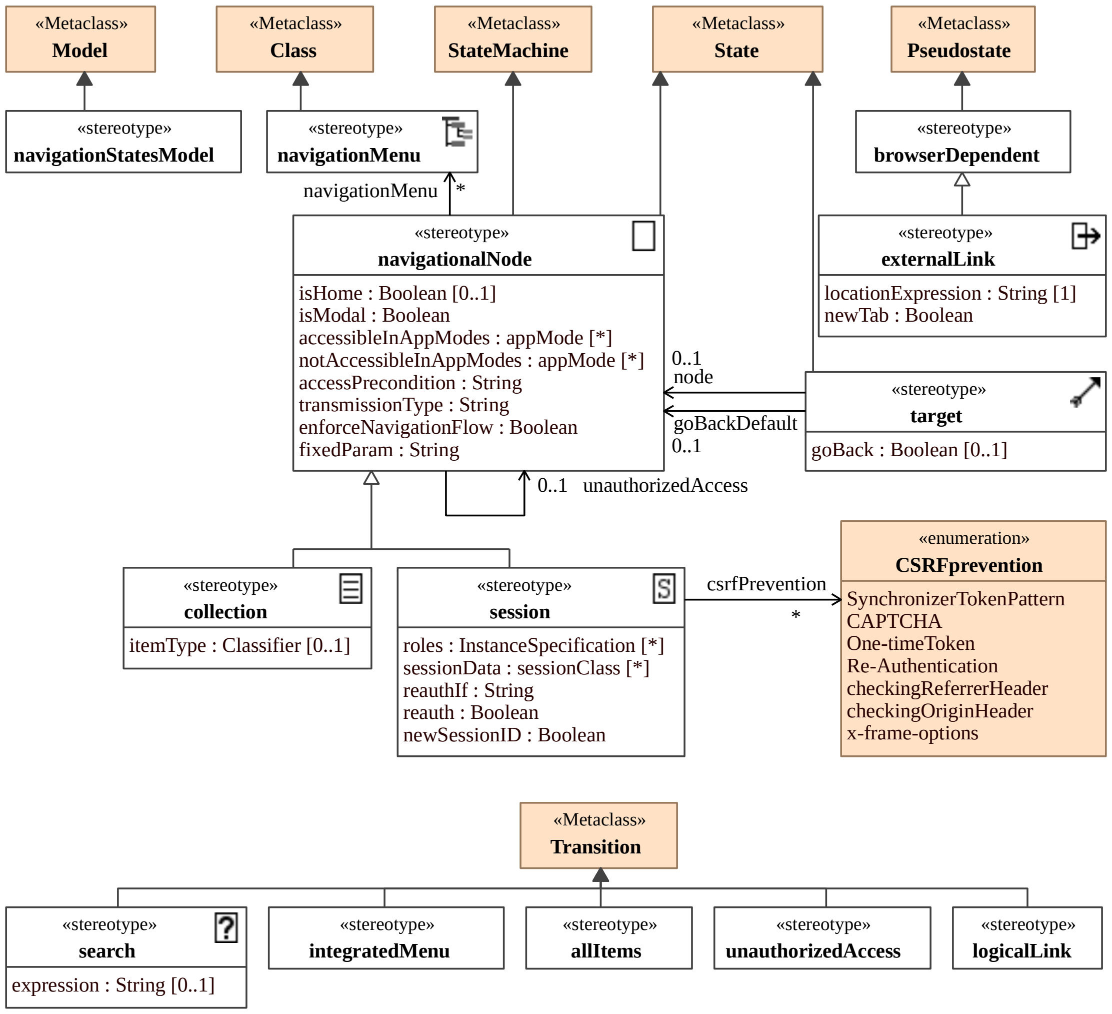


Figure C.5: UWE profile: Navigation model

C.6 Application States Model

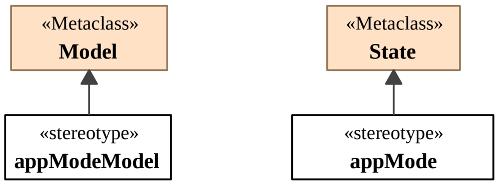


Figure C.6: UWE profile: Application States model

Appendix D

Case Study: Energy Management System

Exemplarily, one of our case studies that we briefly introduced in section 5.1 – namely the Energy Management System (EMS) – is presented in this chapter. We first introduce the surrounding and general requirements, before security-related requirements and design decisions are discussed together with a selection of UWE models of the EMS web application. This chapter is based on joint work with Nora Koch, and Santiago Suppan [44]. All diagrams of our EMS case study can be downloaded from the UWE web site [224].

D.1 Environment and Requirements

This section describes the Energy Management System (EMS) case study and in particular the web application of the EMS that controls Smart Homes, which are households with interconnected appliances. We start by introducing Smart Home components, continue by presenting actors and conclude by explaining concrete functionality, before we go into more security-related details in the next section.

D.1.1 Components of Smart Homes

Figure D.1 depicts a "Smart Home". Generally, the EMS is an interface for the SmartGrid customer that displays consumption data. Concrete instantiations can be realized by an application that provides functionality for energy trading or for regulating the current drain. Ideally, most appliances, as e.g., ovens, dishwashers, washing machines or lamps are so-called Smart Appliances (SAs), which means they contain a small embedded-system that receives control commands from the EMS and that informs the EMS about the current status. Additionally, SAs can be controlled by pushing a button or by using an integrated touch screen.

For a household, exactly one EMS and one Smart Meter are installed locally, in a place where they are protected from physical tampering. The Smart Meter is responsible for monitoring the amount of energy that is sold or bought. As the EMS is connected to the web, remote access to its web application allows users to interact with the EMS and to monitor energy consumption from outside their homes.

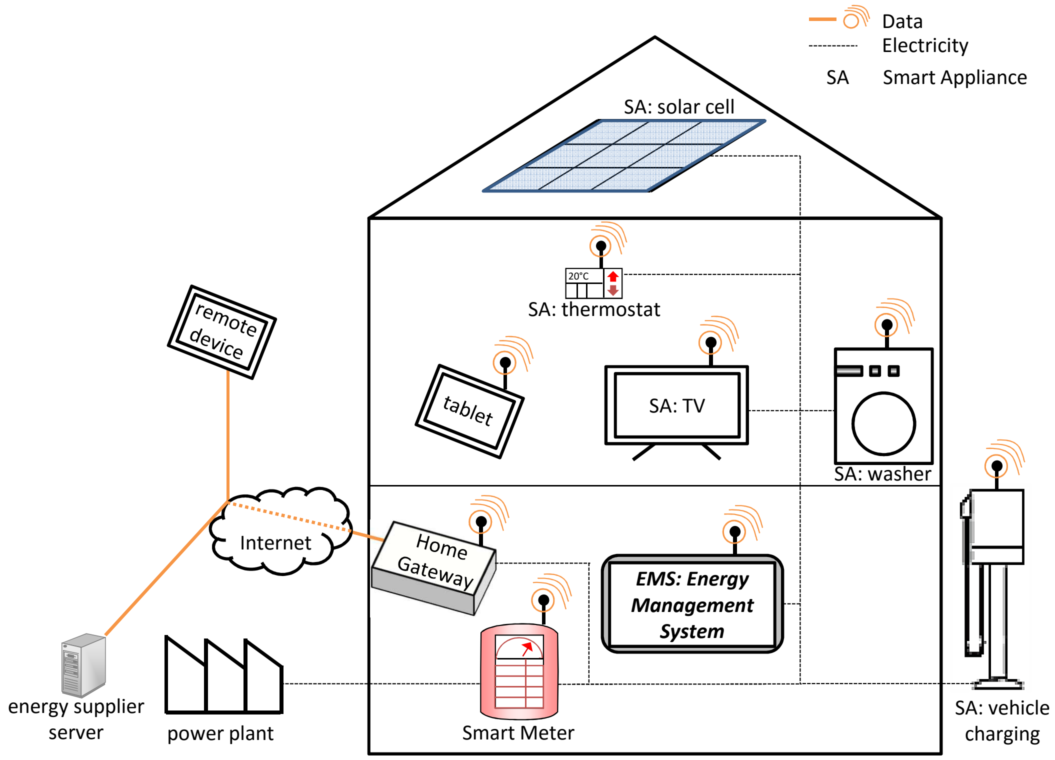


Figure D.1: Entities in the Smart Home (adapted from [65])

A possibility to control energy consumption more globally is Demand Side Management. It envisions adapting the consumption level according to messages sent by energy providers. For example, in situations when lots of energy is needed in an area, the energy provider notifies all Energy Management Systems. Consequently, the EMS can send command messages to SAs in order to turn them off. The concrete behavior when receiving a Demand Side Management message can be controlled by user defined policies in the EMS.

D.1.2 Actors

According to [64], the *pro*sumer (producer / consumer) is the end customer, who is consuming energy as well as producing energy, e.g., by using photovoltaic or wind energy as decentralized energy resources. Prosumers are also able to store energy, for instance in the batteries of the electric vehicle and to resell the energy later to the so-called microgrid[1] when the prices are higher. We also refer to the prosumer simply as “(private) user” or “customer”.

Figure D.2 depicts a UML use case diagram, which gives an overview of the actors in our case study and the main functionalities of the EMS. On the left, the private user is shown. Users can create and configure other users, e.g., under-aged family members can be allowed to sign into the EMS web application and to see their energy consumption, but they should not be able to trigger electricity vending or purchasing functions. On the right, the Meter Point Operator (MPO) is

[1]The term microgrid [96] refers to areas where small communities trade local energy, in addition to the energy supply provided by professional energy suppliers.

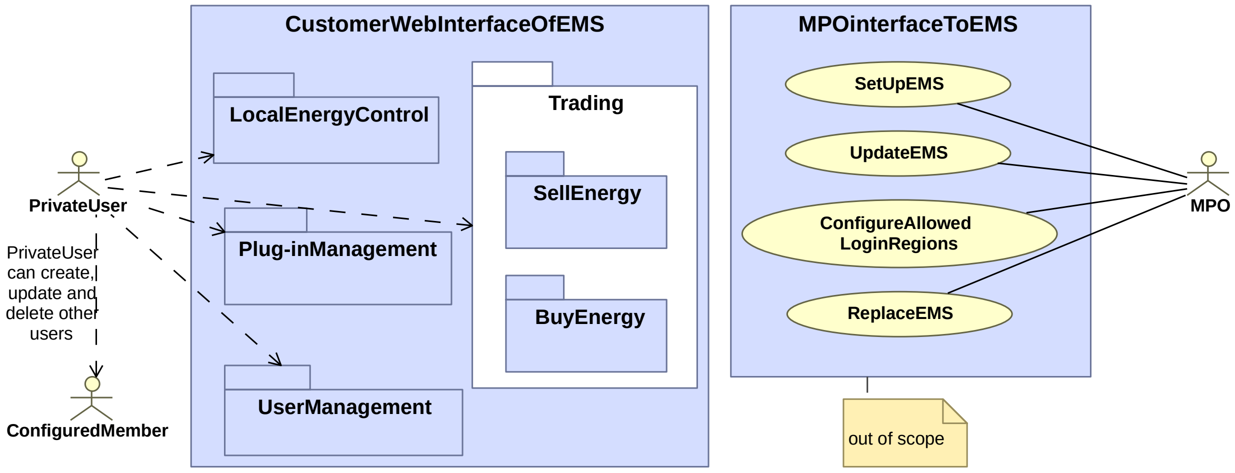


Figure D.2: EMS: Requirements overview

depicted, who is responsible for installing, maintaining or replacing the EMS as well as the Smart Meter. The tasks of the MPO are not considered in our case study.

D.1.3 Functionality

The main functionalities of the EMS are depicted in figure D.2: a user can buy or sell energy, control local energy consumption by configuring SAs, install plugins to automate tasks or manage other users. These use cases are described in more detail in the following.

Local Energy Control

As more and more Smart Appliances will be added to the home network, their heterogeneous functionality has to be made available to the customer. The EMS web application can present, in a uniform way, a coherent view to the user in the form of a portal, presenting information that the EMS has collected from diverse sources (appliances or external servers). SAs, even new ones that were non-existent when the web application was programmed and deployed, offer their services through a standard interface to the EMS (cf. lower half of figure D.3, use case `InteractWithSA`, depicted in bold font because it might be used relatively often). Hereby, auto-configuration (in the sense of plug-and-play support) is important, as many customers may not become acquainted with the full potential of the EMS. This case applies particularly to senior citizens.

Easy access to real-time information supports the users, e.g., to pay attention to their energy consumption, as depicted at the top of figure D.3. Additionally, automatic peak load management provides smart planning for reducing energy consumption. This Smart Planning feature (cf. `Configure Smart Planning Policy`) can be enriched by plugins, which have to be installed separately. Plugins might also be allowed to access the local usage history from SAs. This way they can base their plan on previous user's behavior. For example, hydronic heating might be reduced automatically at times where usually no great quantity of hot water is needed.

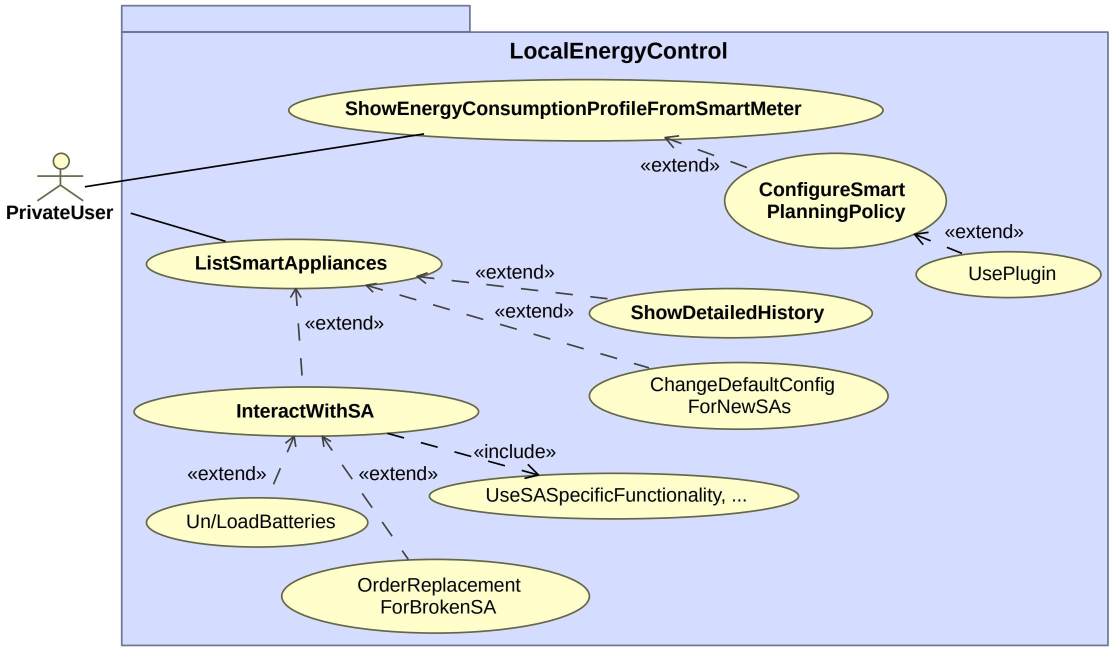


Figure D.3: EMS: Requirements of local energy control

Energy Trading

Selling and buying energy is a critical task, if the user wants the system to perform a trade automatically. Consequently, policies have to ensure that the system acts in the interest of the prosumer. As depicted in figure D.4, the recommendation / trade system can also be enriched by plugins, offering so-called value added services. A value added service, such as a price comparing third party service (e.g., when and who is offering the best conditions for green energy?) works as follows: The third party provides a plugin that obtains current market prices from the third party's server. The plugin compares prices and consumption data locally. The result of the comparison can either be a visual notification in the EMS or a process is started to renegotiate Energy Supplier contracts, in case the prosumer has allowed automatic negotiation. In the latter case, a notification is sent to the user after the (un-)successful provider change.

Plugin Management

As mentioned before, a key functionality is the interplay of the EMS and value added services. Third parties can offer plugins that can be deployed into the EMS to provide further functionality (cf. figure D.5). Plugins are limited, sandboxed algorithms that can enhance the EMS at two predefined interfaces: the interface for smart planning (see local energy control) and/or the interface for energy trading.

The customer accesses the EMS as a central administration point. No process should demand direct interaction between the customer and an external third party service. Users can only search for plugins, (un)install or update them (if not done automatically) or access a privacy dashboard for plugins. The dashboard allows the user to restrict personal information a plugin can access and functions it can execute.

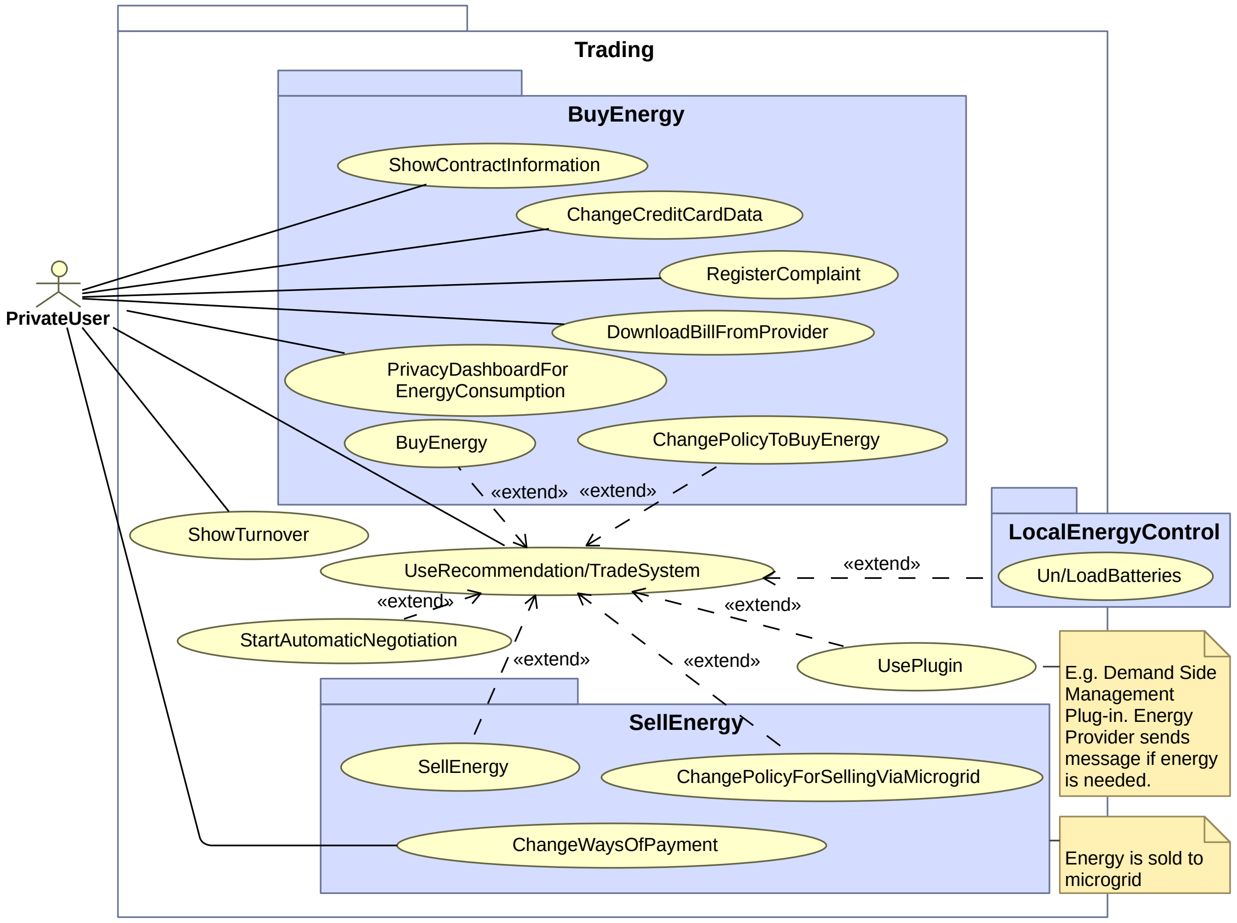


Figure D.4: EMS: Requirements of the energy trading system

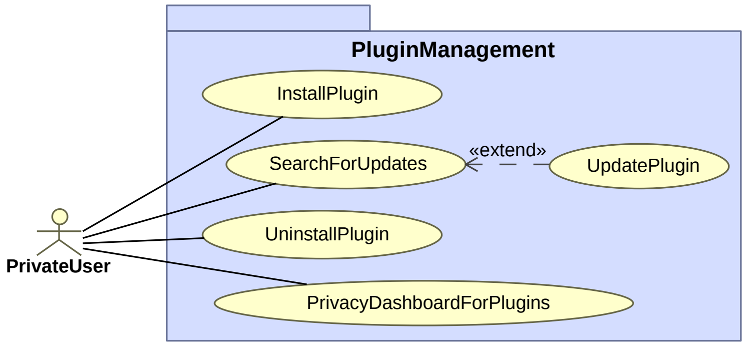


Figure D.5: EMS: Requirements of the plugin management

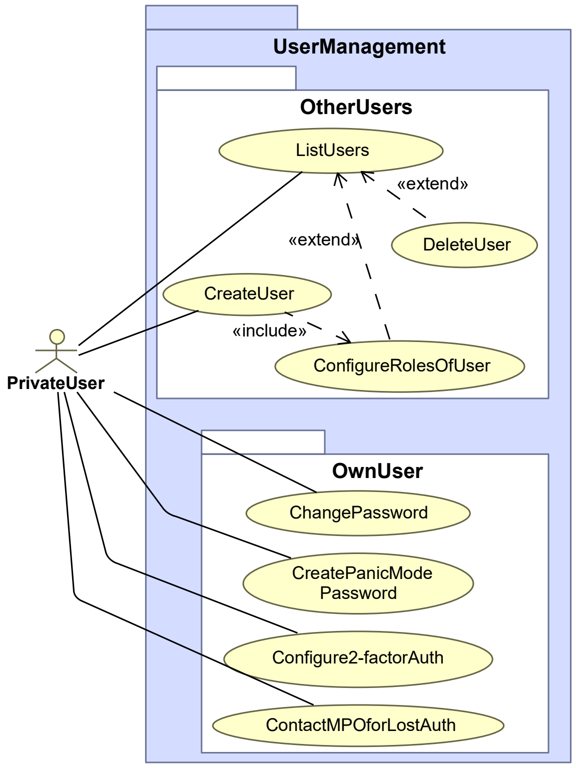


Figure D.6: EMS: Requirements of the user management

User Management

Prosumers can allow other persons to log into the web application by creating an account for them, as depicted in figure D.6. However, not all users have to have the same rights. More details about access control and other security features are given in the next section. Additionally, more comprehensive descriptions of general smart grid functional requirements can be found in [62, 90].

D.2 Securing the EMS Web Application

This section introduces security features of the EMS, i.e., authentication, panic mode, reauthentication, secure connections, authorization, user zone concept, cross-site-request-forgery prevention, under attack mode and SQL-injection prevention (cf. our SecWAO ontology in chapter 4).

Implementing coherent *authentication* is a challenge, as users must be able to log-in to their EMS internally, from their home, and externally, using a mobile device, or a public terminal. A two-factor authentication should be employed to access sensitive information of the EMS. Two-factor authentication requires a knowledge factor ("something only the user knows") and either a possession factor ("something only the user has") or an inherence factor ("something only the user is") from the user for the authentication to succeed. For example, a password has to be entered together with a code that the user's smart phone generates.

A feature rarely implemented in current web applications, is the *panic mode*. When the panic mode is activated, the user interface will be displayed with reasonable information generated by the EMS that does not reflect the user's real information. This is especially needed for coercion situations, where criminals might physically force users to reveal information of themselves or to conclude long-term contracts with certain parties. The panic mode also protects threatened users by pretending to malfunction or to execute functions successfully without any real impact. Therefore, users have to authenticate themselves with predefined credentials which differ from the usual ones: using the same username in combination with a panic mode password loads the alternative user interface.

Besides the first authentication, prosumers can be forced to *reauthenticate* themselves. This is often the case after a certain time of inactivity (often referred to as "automatic logout" in online banking applications), but it is also common for critical areas. For example, web shops often allow to store cookies to keep the user authenticated while browsing their offers. However, if the last authentication is older than a certain amount of time, the users have to reauthenticate themselves before being able to make a purchase. Regarding the EMS plugin installation functions, the last authentication of a prosumer should not be older than 10 minutes, a typical time threshold also used in online banking. The timeout avoids a takeover of a session by another person who has access to the prosumers browser.

All kinds of authentication are useless, if the login process can be eavesdropped. *Secure connections*, as e.g., TLS connections can be used to ensure the confidentiality, integrity and freshness of all user's request as well as of all response of the EMS. As encrypting a connection is a time consuming task, it is an important design decision which parts of an application should be secured. In the case of Energy Management, security weights more than speed, even if Demand Side Management and energy trading are very time demanding [17]. Compromises in speed can have impact on economic aspects, but compromises in security could mean a total blackout of the power supply, producing high economic damages.

Apart from secure session management after authentication, a well implemented *authorization (access control)* concept is needed to satisfy customer needs. There are several roles to be considered, as family members might be involved in the customization of the Smart Home.

Many web applications require a *user zone concept*. If users are accessing the EMS from the home area, they are permitted to access all prosumer managing functions (depending on their roles). But if they are requesting access externally, stricter policies have to be enforced, depending on the requester's location, i.e. the IP address of the requester's device. To configure this policy, users inform their MPO that they are on holiday and that a certain location is the source of legitimate requests.

A telling example is an attack from a foreign country. An attacker that is mimicking a user will, by policy enforcement, be denied to alter the Smart Appliances' behavior, if he is accessing the EMS remotely from a very far place. This feature will not hold up against versatile attackers, as several proxies or even computers that have been compromised by an attacker, could be available in the desired geo-location. Still, this mechanism represents a filter against unambitious attackers. There are several other mature attacks on web based technologies that also could have an impact on the EMS, mostly related to so-called "common web application vulnerabilities" [161]. As the EMS is remotely accessible by means of a web client, there is room for session riding attacks. Depending on the user's browsing application, *cross-site-request-forgery* (abbreviated "CSRF") might be used by a malicious attacker to trigger actions without the user's consent. For example, an attacker could trick users into interacting with the web server of a Smart Appliance by letting them call an address like:

`http://EMSremoteIP.com/SmartApplianceName/SmartApplianceFunction`
This request cannot be called by an unauthorized person due to the policy enforcement inside the EMS, but it can be triggered by means of CSRF.

The *under attack mode* is a dynamic protection against attempts of compromising the EMS functionality, as the EMS reacts accordingly and reduces the attackers possibilities. An example is the reduced functionality when under denial of service attack. The EMS will try to reduce the number of allowed connections and/or deny any connection from IPs that have exceeded a certain number of requests in a certain time frame. Additionally, CAPTCHA-challenges could be displayed to verify that the requester is a person and not merely a program.

Another feature is *the protection of the EMS database.* The EMS database should only accept statements that have been generated by the EMS itself. In order to avoid SQL-injection attacks within generated statements, parameterized queries should be used.

Finally, *secure downloads* are needed for firmware updates from the MPO and for downloading bills from an Energy Supplier. The document representing the bill should be available in the trading system and prosumers should be able to download the bill so that its integrity is preserved. Naturally, confidentiality and integrity are required for the bill's storage.

D.3 Modeling the EMS with UWE

This section shows how to model security features of the EMS with UWE model elements. It is structured according to the UWE model types, i.e., the views on the web application.

D.3.1 Content View

When modeling larger systems, such as the EMS web application, it is useful to divide the system into manageably small components. The main characteristic of components is encapsulation, which means that components can only share information using predefined interfaces. Encapsulation is advantageous, because each component can be implemented and tested individually. Regarding modeling, components contribute to a clear structure, as the division of tasks within an application becomes apparent. Consequently, it is easy to define appropriate security properties for each part of a web application.

In the case of our EMS, a component `EMScore` is created, which contains components that are built into the EMS system by default, as depicted in the class diagram shown in figure D.7. Smart Appliances (SAs) can communicate with the EMS using the `SA` interface, shown on the lower left. According to the description in the previous section, plugins are also external components that can enhance the smart planning or the trader / recommender. Some plugins might provide both functionalities (as e.g., `PluginA` does).

The `EMScore` contains four internal components that correspond to the main areas we identified in the requirements phase (cf. figure D.2): local energy control, user management, energy trade system and plugin management. As can be seen in figure D.7, the user manager is used by all components, because the system does not allow access without having granted permission first. The interface `PluginList` publishes the list of installed plugins within the system so that the user can advise the internal components to exchange the planning or trading plugin.

As far as security is concerned, the UWE profile redefines the UML stereotype ≪component≫ with the following tags that have already been defined in section 5.3:

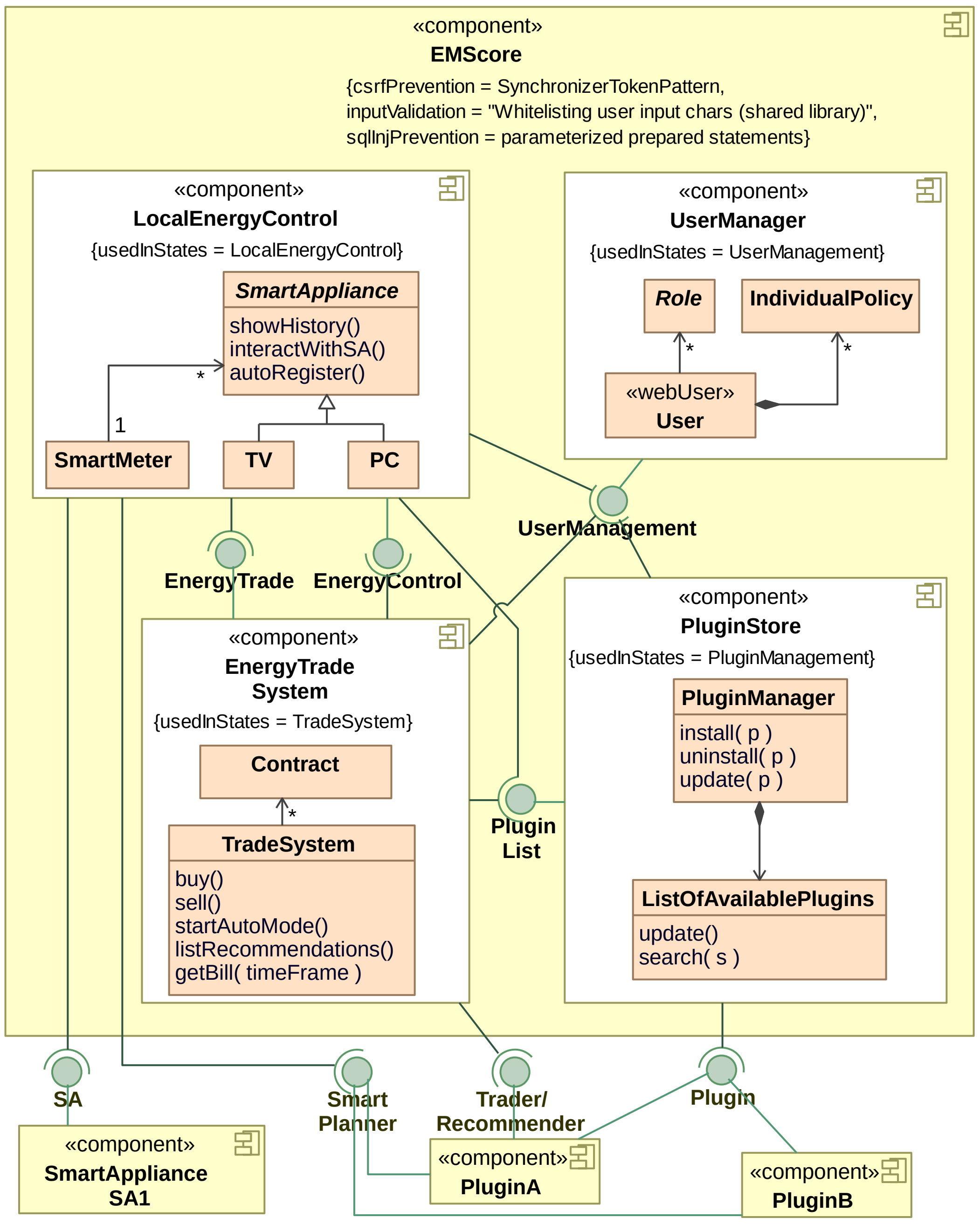


Figure D.7: EMS: Content model

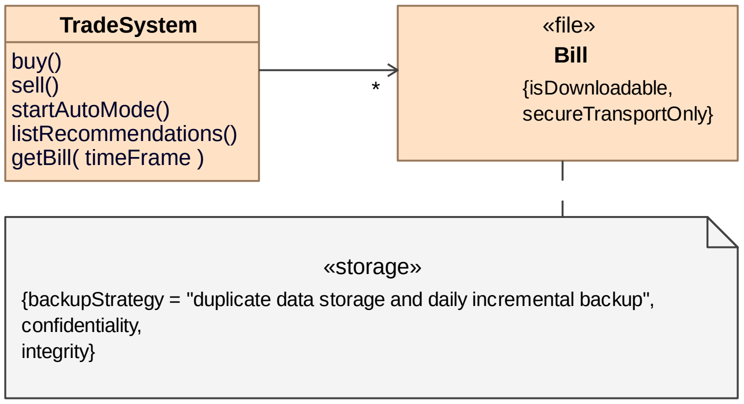


Figure D.8: EMS: Content model – bills

{csrfPrevention} models how Cross-Site-Request-Forgery (CSRF) should be repelled. The modeler can choose from the options presented at the OWASP CSRF Cheat Sheet [164]. For the EMS example the most common "Synchronizer Token Pattern" is used, which includes a randomly generated challenge token to all server requests in a web page. An attacker cannot hope to guess a valid token when sending the user a prepared URL.

{sqlInjPrevention} documents how SQL injection attacks are prevented. In most programming languages, SQL prepared statements shield from SQL injection, but other solutions, as e.g., server-sided stored procedures could also be used.

{inputValidation} explains how the component is shielded from unvalidated input. The most secure way is to whitelist characters and not to accept anything else. In a later phase of development, it could be useful to use this tag for documenting the concrete technique which is used, e.g., a software library.

The tag {usedInStates} is used to denote in which state of the application a certain component is used. Note that it is the modelers' decision which of the features offered by the UWE profile they like to use in a diagram. In some scenarios, the modelers may decide to connect the UWE Navigation model with the Content model using {usedInStates}, in others not.

Figure D.8 exemplarily depicts the storage and the download of a bill, using UWE's ≪storage≫ tag on a UML comment and the ≪file≫ stereotype to specify that a bill can be downloaded, but only over a secure connection that preserves integrity, as e.g., TLS.

D.3.2 Role and Access Control View

Figure D.9 depicts the role model of the EMS. The stereotype ≪webUser≫ defines the class that represents a user. It can later be referred to as `caller` when defining access control. Per default, the `DefaultUser` plays all roles.

For our EMS application, figure D.10 shows an excerpt of the static Basic Rights model[2]. For example, someone with the role `UserManager` is allowed to ≪delete≫ users, as long as the

[2]In contrast, figure 5.4 in section 5.2 depicted a dynamic view, i.e., individual access control specified by the users themselves.

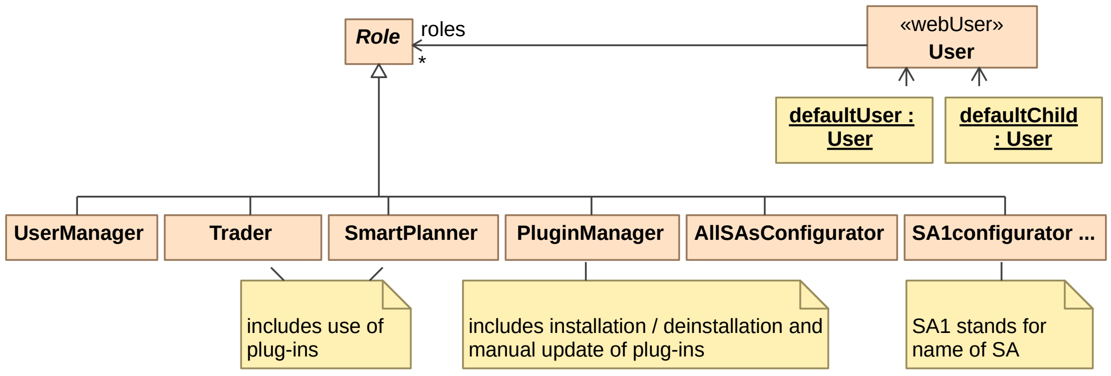


Figure D.9: EMS: Role model

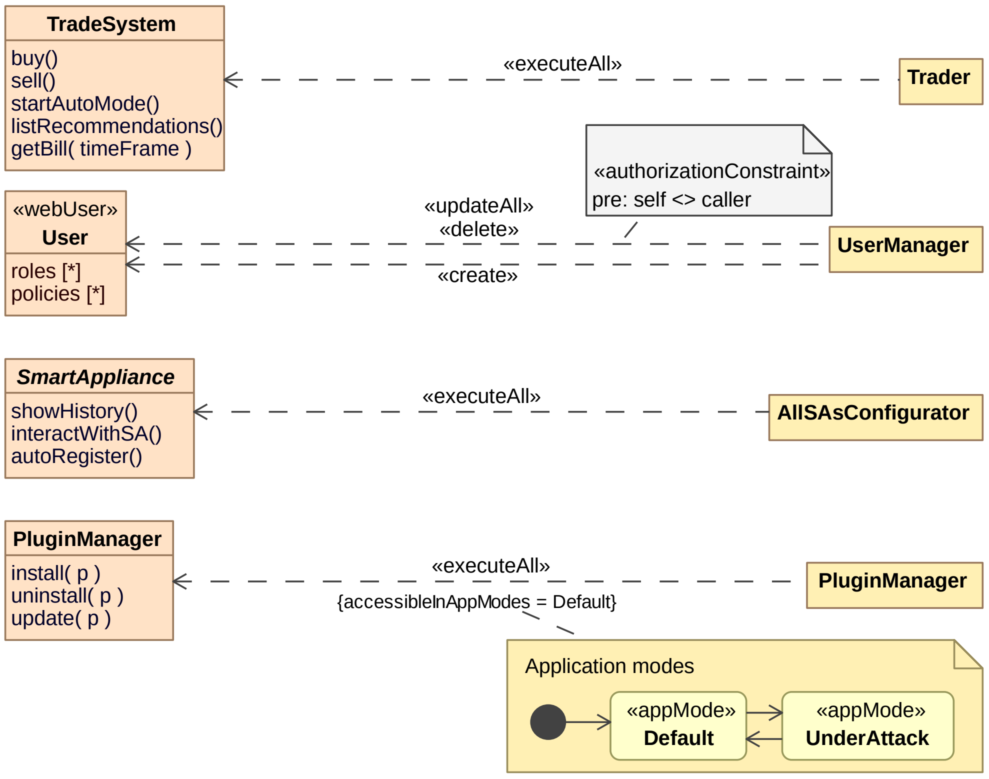


Figure D.10: EMS: Basic Rights model excerpt and Application States model

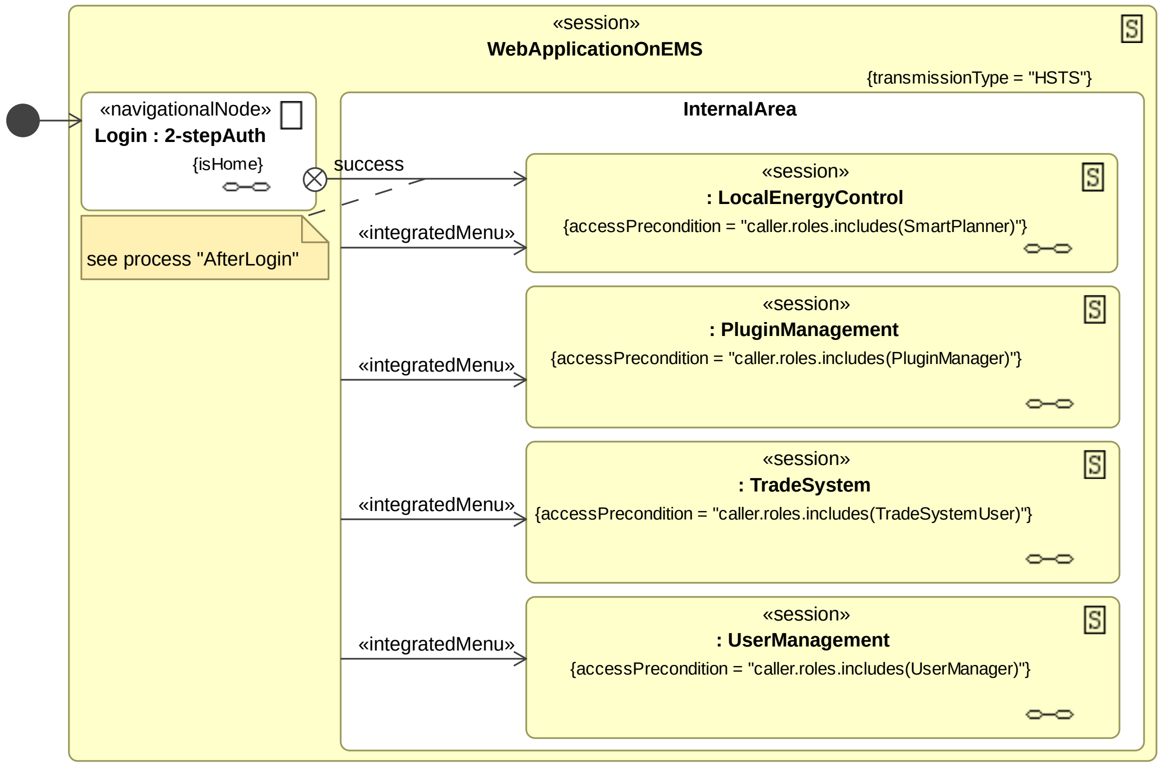


Figure D.11: EMS: Navigation model overview

≪authorizationConstraint≫, which has to be specified in OCL [151], is fulfilled. The constraint defines that the user instance (referred to as `self`) is not equal to the `caller` (referring to the ≪webUser≫ who executes this deletion). If the constraint would not be given, users might delete their own accounts accidentally.

Regarding our security requirements, the UWE Basic Rights model has to be extended to enable the specification of different modes. Therefore, the tags {accessibleInAppModes} and {notAccessibleInAppModes} are added. They allow to choose from a set of states in which a functionality should (not) be available. Please note that these states do not refer to navigational states, but general states of an application. When modeling with a CASE tool as MagicDraw [147], the UWE profile with its typed tags makes sure that the value for the tag can only be chosen from all available states that are stereotyped by ≪appMode≫. As depicted at the bottom of figure D.10, the EMS only allows the installation of plugins when it is not under attack.

D.3.3 Navigation and Process View

User navigation is one of the most distinguished web features. UML state charts are used in UWE to express the navigation possibilities a user has within a certain state of the web application [39]. By default, all states in the UWE navigation model are thought to be stereotyped ≪navigationalNode≫. The {isHome} tag refers to the entry point of a web application (cf. figure D.11).

The stereotype ≪integratedMenu≫ is defined to be a shortcut for showing menu items for all menus of Submachine States, in case the user is allowed to access them [37]. Submachine

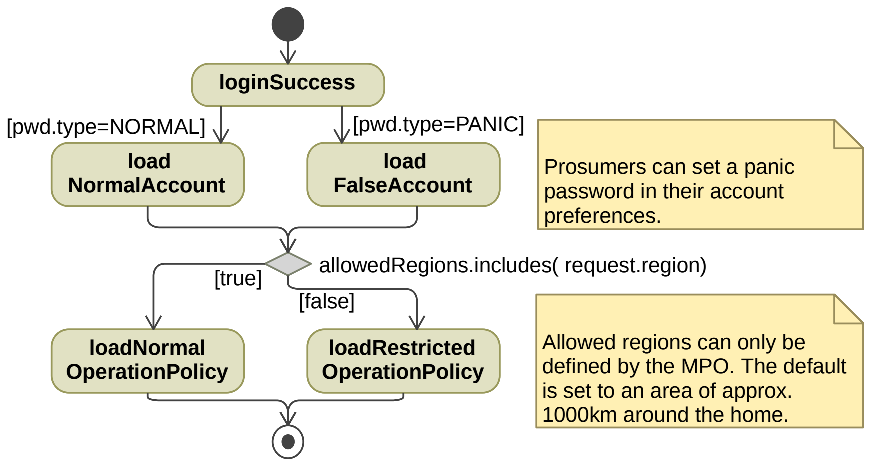

loginSuccess
[pwd.type=NORMAL]
[pwd.type=PANIC]
load NormalAccount
load FalseAccount
Prosumers can set a panic password in their account preferences.
allowedRegions.includes(request.region)
[true]
[false]
loadNormal OperationPolicy
loadRestricted OperationPolicy
Allowed regions can only be defined by the MPO. The default is set to an area of approx. 1000km around the home.

Figure D.12: EMS: Process after successful login

states contain a state machine by themselves, so that more details can be shown in another diagram. Note that transitions that start at the border of a state, leave the state and enter again when triggered. Navigational access control can be specified using a `rolesExpression` as “caller.roles.includes(PluginManager)”.

As shown on the left in figure D.11, a UWE pattern is used for the specification of 2-step authentication. The UWE profile includes such kind of patterns to reduce the amount of modeling effort. In case a pattern should be adapted, it can easily be copied from the profile to the model.

Additionally, we decided to include more implementation-specific details in the navigation model, which are relevant in the late design phase. Thus, HTTP Strict Transport Security (HSTS) is specified as web security policy mechanism to ensure secure https connections for the whole web application, indicated by the ≪session≫-related tag {transmissionType=HSTS}.

If needed, activity diagrams can be added to detail the process that is executed behind the scenes. For example, figure D.12 depicts what happens internally after the login was completed successfully. For our EMS, this gives a hint to implement the panic mode as well as the restricted access, when accessing the EMS from a distant region.

Exemplarily, the submachine state diagram of the plugin management is shown in figure D.13. The stereotype ≪search≫ denotes that a search is done when using the `searchPlugins` transition, as searching is a typical process in applications. The stereotype ≪collection≫ refers to a list of elements with the given {itemType} tag from the Content model. For transitions, an underscore can be used to denote an element of this type. In our example, the underscore is an abbreviation for `p:Plugin`.

The UWE profile provides a new tag called {reauth} for the stereotype ≪session≫ to specify critical areas. In those areas, as e.g., for plugin management, users have to reauthenticate themselves, except when the previous login is not older than the given amount of time. In addition, the tag {notAccessibleInAppModes} is specified for the stereotype ≪navigationalNode≫. In our example, this prevents navigating to the interface for (un)installing or updating plugins in the `UnderAttack` mode.

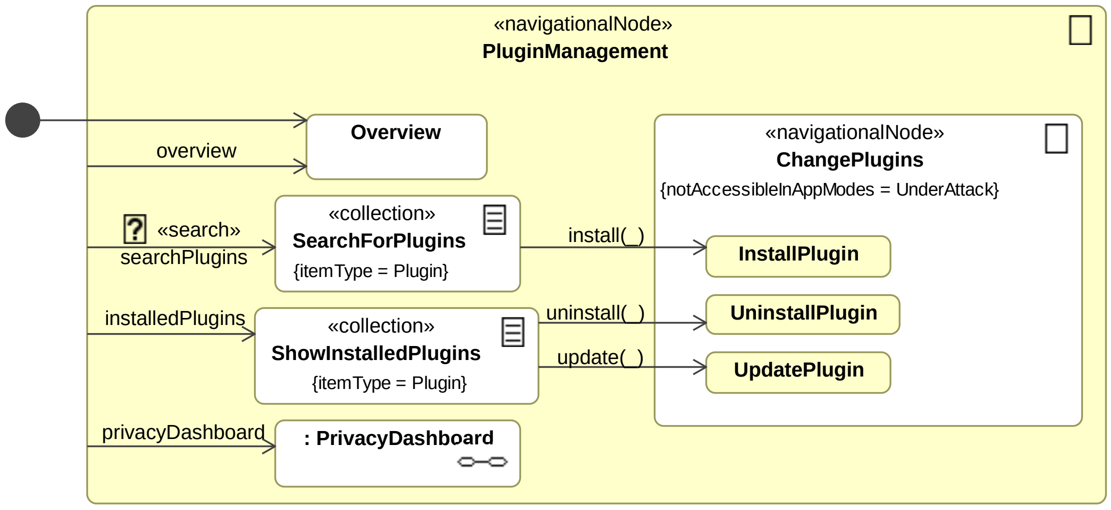


Figure D.13: EMS: Navigation model for plugin management

Publications of Marianne Busch

This chapter enumerates the author's publications. For each joint publication, the author's contribution is described. For publications quoted in this thesis, a reference to the bibliography is provided. In case a chapter is based on a publication, a link to the chapter is added.
This thesis can be downloaded from http://www.pst.ifi.lmu.de/~busch/thesisMarianneBusch.pdf

2015

M. Busch and M. Wirsing: An Ontology for Secure Web Applications. In Ruqian Lu, editor, International Journal of Software and Informatics, volume 9 of International Journal of Software and Informatics, pages 233–258. Institute of Software, Chinese Academy of Sciences, 2015. (peer reviewed, 26 pages. Cited as [48] in chapter 4. M. Busch's contribution: ABox SecWAO and further development of SecEval's Security Context model to serve as a TBox for this ontology.)

2014

M. Busch, N. Koch, and S. Suppan: Modeling Security Features of Web Applications. In M. Heisel, W. Joosen, J. Lopez and F. Martinelli (eds.) *Engineering Secure Future Internet Services.* LNCS, vol. 8431, ISBN:978-3-319-07451-1. Springer, pp. 119–139, `doi:10.1007/978-3-319-07452-8_5`. (21 pages. Cited as [44] in appendix D. M. Busch's contribution: UWE extension for security, e.g., regarding SQL-injection prevention, cross-site-request-forgery prevention, authentication, reauthentication, secure connections, under attack mode, authorization, user zone concept, panic mode. UWE diagrams and descriptions of the presented case study.)

M. Busch, N. Koch, and M. Wirsing: Evaluation of Engineering Approaches in the Secure Software Development Life Cycle. In M. Heisel, W. Joosen, J. Lopez and F. Martinelli (eds.) *Engineering Secure Future Internet Services.* LNCS, vol. 8431, ISBN:978-3-319-07451-1. Springer, pp. 234–265, `doi:10.1007/978-3-319- 07452-8_10`. (32 pages. Cited as [45] in chapter 3. M. Busch's contribution: Extended version of our work presented in "Modellierung 2014". SecEval in greater detail, including two extensions and full case study.)

A. Bertolino, M. Busch, S. Daoudagh, F. Lonetti, and E. Marchetti: A Toolchain for Designing and Testing Access Control Policies. In M. Heisel, W. Joosen, J. Lopez and F. Martinelli (eds.) *Engineering Secure Future Internet Services.* LNCS, vol. 8431, ISBN:978-3-319-07451-1. Springer, pp. 266–286, `doi:10.1007/978-3-319-07452-8_11`. (21 pages.

Cited as [20] in section 6.3. M. Busch's contribution: Toolchain orchestration in the Service Development Environment (SDE). All UWE-related tools of the toolchain, i.e., "editProjectWithMagicUWE", "transformUWE2xacml", "checkConsistency", which are three out of six tools in the toolchain.)

M. Busch and N. Koch et al. Deliverable for EU-Project NESSoS: D2.5 – Third Release of the Method and Tool Evaluation. (88 pages. M. Busch's contribution: Motivating tool owners/users to adding their tools to the SDE and to describe their experience with the integrated tools. Tool integrations are often joint work. Overview of contributions for the NESSoS project in work package 2, i.e., overview of SecEval and the SDE with set of 25 integrated tools and two toolchains.)

M. Busch, N. Koch, and M. Wirsing. SecEval: An Evaluation Framework for Engineering Secure Systems. In H. Fill, D. Karagiannis, and U. Reimer, editors, *Modellierung 2014*. Gesellschaft für Informatik e. V. (GI), pp. 337–352. (peer reviewed, 16 pages. Cited as [46] in chapter 3. M. Busch's contribution: SecEval. Diagrams and descriptions of the presented case study.)

M. Busch. Secure Web Engineering supported by an Evaluation Framework. In *Doctoral Consortium, Modelsward Conference 2014*, Scitepress. Best PhD Student award. (peer reviewed, 8 pages, 2 columns)

2013

A. Bertolino, M. Busch, S. Daoudagh, N. Koch, F. Lonetti, and E. Marchetti. A Toolchain for Designing and Testing XACML Policies. In *Proceedings of ICST 2013*. IEEE, pp. 495–496, `doi:10.1109/ICST.2013.70`. (peer reviewed, 2 pages, 2 columns & poster. M. Busch's contribution: Parts regarding modeling access control and model transformation in jointly envisioned toolchain.)

M. Busch, M. Ochoa, and R. Schwienbacher. Modeling, Enforcing and Testing Secure Navigation Paths for Web Applications. Technical Report 1301, Ludwig-Maximilians-Universität München. (17 pages. Cited as [47] in section 6.5. M. Busch's contribution: Initial idea of navigation paths in discussion with Martín Ochoa, further elaboration in the context of the supervision of R. Schwienbacher's bachelor's thesis.)

M. Busch and N. Koch et al. Deliverable for EU-Project NESSoS: D2.4 – Second Release of the Method and Tool Evaluation. (76 pages. Cited as [42]. M. Busch's contribution: Background description, SecEval and case study represented as SecEval models.)

2012

M. Busch and M. García de Dios. ActionUWE: Transformation of UWE to ActionGUI Models. Technical Report 1203, Ludwig-Maximilians-Universität München. (30 pages. Cited as [38] in section 6.4. M. Busch's contribution: Adaptation of UWE so that it can express, e.g., necessary links between models – the transformation itself is joint work.)

M. Busch, N. Koch, M. Masi, R. Pugliese, and F. Tiezzi. Towards model-driven development of access control policies for web applications. In *1st Workshop on Model-Driven Security (MDsec 2012)* on MoDELS 2012, LNCS. ACMDL, pp. 41–46, doi:10.1145/2422498.2422502. (peer reviewed, 6 pages, 2 columns. Cited as [43] in section 6.2. M. Busch's contribution: Transformation from UWE's Basic Rights model to XACML and the updated HospInf case study.)

M. Busch and N. Koch et al. Deliverable for EU-Project NESSoS: D2.3 – Second release of the SDE for Security-Related Tools. (65 pages. M. Busch's contribution: Extending the SDE. Integrating, e.g., UWE2XACML. Usage example for the integrated use of MagicUWE in the SDE. Motivating tool owners/users to adding their tools and usage examples to the SDE and overseeing the integration of 13 tools in total.)

2011

M. Busch, A. Knapp, and N. Koch. Modeling Secure Navigation in Web Information Systems. In J. Grabis and M. Kirikova, editors, *10th Int. Conf. on Business Perspectives in Informatics Research*, LNBIP. Springer, pp. 239–253, doi:10.1007/978-3-642-24511-4_19. (peer reviewed, 15 pages. Cited as [39]. M. Busch's contribution: Abridging and revising, as this paper publishes content of the author's Diplomarbeit.)

M. Busch and N. Koch et al. Deliverable for EU-Project NESSoS: D2.1 – First Release of Method and Tool Evaluation (46 pages. Cited as [41]. M. Busch's contribution: Evaluation of three methods, three notations, and three tools, according to the first version of CBK's metamodel. The metamodel and its relations is joint work with the University of Duisburg-Essen. Overview of Knowledge Objects in the NESSoS project in relation to the SDLC.)

M. Busch and N. Koch et al. Deliverable for EU-Project NESSoS: D2.2 – First Release of the SDE for Security-Related Tools. (45 pages. M. Busch's contribution: Integrating, e.g., MagicDraw into the SDE. Updating and adapting SDE's usage tutorial. Motivating tool owners/users to adding their tools and usage examples to the SDE and overseeing the integration of 8 tools.)

M. Busch. Integration of Security Aspects in Web Engineering. Diplomarbeit (Master's Thesis), Ludwig-Maximilians-Universität München. (145 pages. Cited as [37].)

2009

M. Busch and N. Koch. MagicUWE – A CASE Tool Plugin for Modeling Web Applications. In M. Gaedke, M. Grossniklaus, and O. Díaz, editors, *Proc. 9th Int. Conf. Web Engineering (ICWE'09)*, LNCS 5648. Springer, pp. 505–508, doi:10.1007/978-3-642-02818-2_49. (peer reviewed, 4 pages + poster. Cited as [40]. M. Busch's contribution: Refactoring a prototype of a MagicDraw plugin and implementing new features for modeling UWE with our MagicUWE plugin.)

M. Busch and N. Koch. Rich Internet Applications. State-of-the-Art. Technical Report 0902, Ludwig-Maximilians-Universität München. (18 pages. M. Busch's contribution: Investigating current definitions of Rich Internet Applications (RIAs) and web frameworks that support RIAs.)

Bibliography

[1] A. Barth. HTTP State Management Mechanism. Version 1.2. Specification, Internet Engineering Task Force (IETF), 2011. https://tools.ietf.org/html/rfc6797.

[2] Acunetix. Web application vulnerability report 2015. Technical report, Acunetix Ltd., 2015. https://www.acunetix.com/acunetix-web-application-vulnerability-report-2015/.

[3] D. Akhawe, A. Barth, P. E. Lam, J. Mitchell, and D. Song. Towards a formal foundation of web security. In *Computer Security Foundations Symposium (CSF), 2010 23rd IEEE*, pages 290–304. IEEE, 2010.

[4] M. Alalfi, J. Cordy, and T. Dean. Recovering role-based access control security models from dynamic web applications. In M. Brambilla, T. Tokuda, and R. Tolksdorf, editors, *Web Engineering*, volume 7387 of *LNCS*, pages 121–136. Springer, 2012. `doi:10.1007/978-3-642-31753-8_9`.

[5] ANSI/HL7. Hl7 Version 3 Standard: Security and privacy ontology, release 1. Technical report, Health Level Seven International, 2014. http://www.hl7.org/implement/standards/product_brief.cfm?product_id=348.

[6] G. Aragón, M.-J. Escalona, M. Lang, and J. R. Hilera. An analysis of model-driven web engineering methodologies. *International Journal of Innovative Computing, Information and Control*, 9(1):413–436, 2012. http://www.ijicic.org/ijicic-11-11012.pdf.

[7] ASCENS Project. Autonomic Service Component Ensembles, 2012. http://www.ascens-ist.eu/.

[8] R. Auger. Insufficient Process Validation, WASC Threat Classification, 2009. http://projects.webappsec.org/w/page/13246943/Insufficient%20Process%20Validation.

[9] D. Basin, M. Clavel, M. Egea, M. de Dios, and C. Dania. A model-driven methodology for developing secure data-management applications. *Software Engineering, IEEE Transactions on*, 40(4):324–337, 2014. `doi:10.1109/TSE.2013.2297116`.

[10] D. Basin, M. Clavel, J. Doser, and M. Egea. A metamodel-based approach for analyzing security-design models. In G. Engels, B. Opdyke, D. Schmidt, and F. Weil, editors, *Model Driven Engineering Languages and Systems*, volume 4735 of *LNCS*, pages 420–435. Springer, 2007. `doi:10.1007/978-3-540-75209-7_29`.

[11] D. Basin, M. Clavel, and M. Egea. Automatic Generation of Smart, Security-Aware GUI Models. In *Engineering Secure Software and Systems*, volume 5965 of *LNCS*, pages 201–217. Springer, 2010.

[12] D. Basin, M. Clavel, M. Egea, M. A. G. de Dios, C. Dania, G. Ortiz, and J. Valdazo. Model-Driven Development of Security-Aware GUIs for Data-Centric Applications. In A. Aldini and R. Gorrieri, editors, *Foundations of Security Analysis and Design VI - FOSAD Tutorial Lectures*, volume 6858 of *LNCS*, pages 101–124. Springer, 2011. doi:10.1007/978-3-642-23082-0_4.

[13] D. Basin, P. Schaller, and M. Schläpfer. *Applied Information Security: A Hands-on Approach.* Springer, 2011. doi:10.1007/978-3-642-24474-2.

[14] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice.* Addison-Wesley Professional, 3 edition, 2012.

[15] K. Beckers, S. Eicker, M. Heisel, and W. Schwittek. NESSoS Deliverable D5.2 – Identification of Research Gaps in the Common Body of Knowledge, 2012. http://www.nessos-project.eu/media/deliverables/y2/NESSoS-D5.2.pdf.

[16] S. Beji and N. El Kadhi. Security ontology proposal for mobile applications. In *Mobile Data Management: Systems, Services and Middleware, 2009. MDM '09. Tenth International Conference on*, pages 580–587, 2009. doi:10.1109/MDM.2009.100.

[17] C. Bennett and S. Wicker. Decreased time delay and security enhancement recommendations for AMi smart meter networks. In *Innovative Smart Grid Technologies (ISGT)*, pages 1–6, 2010. doi:10.1109/ISGT.2010.5434780.

[18] A. Bertolino, S. Daoudagh, F. Lonetti, and E. Marchetti. The X-CREATE framework: a comparison of XACML policy testing strategies. In *Proceedings of 8th International Conference on Web Information Systems and Technologies (WEBIST)*, pages 18–21, 2012.

[19] A. Bertolino, F. Lonetti, and E. Marchetti. Systematic XACML Request Generation for Testing Purposes. In *Proc. of the 36th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*, pages 3–11, 2010.

[20] A. Bertolino, M. Busch, S. Daoudagh, F. Lonetti, and E. Marchetti. A Toolchain for Designing and Testing Access Control Policies. In Heisel et al. [100], pages 266–286. doi:10.1007/978-3-319-07452-8.

[21] A. Bertolino, S. Daoudagh, F. Lonetti, and E. Marchetti. Automatic XACML Requests Generation for Policy Testing. In *Proceedings of IEEE Fifth International Conference on Software Testing, Verification and Validation (ICST)*, pages 842–849, 2012.

[22] A. Bertolino, S. Daoudagh, F. Lonetti, E. Marchetti, and L. Schilders. Automated testing of extensible access control markup language-based access control systems. *IET Software*, 7(4):203–212, 2013.

[23] A. Białas. Semiformal Approach to the IT Security Development. *International Conference on Dependability of Computer Systems*, 0:3–10, 2007. doi:10.1109/DEPCOS-RELCOMEX.2007.43.

[24] A. Białas. Ontology based model of the common criteria evaluation evidences. *Theoretical and Applied Informatics*, 25, 2013.

[25] M. Bishop. *Computer Security: Art and Science*. Addison-Wesley Professional, 1st edition, 2002.

[26] C. Blanco, J. Lasheras, E. Fernández-Medina, R. Valencia-García, and A. Toval. Basis for an integrated security ontology according to a systematic review of existing proposals. *Computer Standards & Interfaces*, 33(4):372–388, 2011. http://www.sciencedirect.com/science/article/pii/S0920548911000043, doi:10.1016/j.csi.2010.12.002.

[27] B. Boehm and V. R. Basili. Software defect reduction top 10 list. *Computer*, 34(1):135–137, 2001. doi:10.1109/2.962984.

[28] D. Boneh, E. Shen, and B. Waters. Strongly unforgeable signatures based on computational diffie-hellman. In M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, editors, *Public Key Cryptography - PKC 2006*, volume 3958 of *LNCS*, pages 229–240. Springer, 2006. doi:10.1007/11745853_15.

[29] P. Bourque, R. E. Fairley, et al. *Guide to the Software Engineering Body of Knowledge (SWEBOK): Version 3.0*. IEEE Computer Society Press, 2014. http://www.swebok.org/.

[30] M. Brambilla and P. Fraternali. Large-scale Model-Driven Engineering of web user interaction: The WebML and WebRatio experience. *Science of Computer Programming*, 2013. doi:10.1016/j.scico.2013.03.010.

[31] B. Braun, P. Gemein, H. P. Reiser, and J. Posegga. Control-flow integrity in web applications. In *ESSoS*, pages 1–16, 2013.

[32] R. Breu. Ten principles for living models-a manifesto of change-driven software engineering. In *Complex, Intelligent and Software Intensive Systems (CISIS), 2010 International Conference on*, pages 1–8. IEEE, 2010.

[33] R. Breu, G. Popp, and M. Alam. Model based development of access policies. *International Journal on Software Tools for Technology Transfer*, 9(5-6):457–470, 2007. doi:10.1007/s10009-007-0045-y.

[34] J. Bryans. Reasoning about XACML Policies using CSP. In *SWS*, pages 28–35. ACM, 2005.

[35] J. Bryans and J. S. Fitzgerald. Formal Engineering of XACML Access Control Policies in VDM++. In *ICFEM*, volume 4789 of *LNCS*, pages 37–56. Springer, 2007.

[36] M. Buchler, J. Oudinet, and A. Pretschner. SPaCiTE – Web Application Testing Engine. *Software Testing, Verification, and Validation, 2008 International Conference on*, 0:858–859, 2012. doi:10.1109/ICST.2012.187.

[37] M. Busch. Integration of Security Aspects in Web Engineering. Master's thesis, Ludwig-Maximilians-Universität München, 2011. http://uwe.pst.ifi.lmu.de/publications/BuschDA.pdf.

[38] M. Busch and M. A. García de Díos. ActionUWE: Transformation of UWE to ActionGUI Models. Technical Report 1203, Ludwig-Maximilians-Universität München, 2012.

[39] M. Busch, A. Knapp, and N. Koch. Modeling Secure Navigation in Web Information Systems. In J. Grabis and M. Kirikova, editors, *10th International Conference on Business Perspectives in Informatics Research*, volume 90 of *LNBIP*, pages 239–253. Springer, 2011. doi:ACASEToolPluginforModelingWebApplications.

[40] M. Busch and N. Koch. MagicUWE — A CASE Tool Plugin for Modeling Web Applications. In *Proc. 9th Int. Conf. Web Engineering (ICWE'09)*, volume 5648 of *LNCS*, pages 505–508. Springer, 2009. doi:10.1007/978-3-642-02818-2_49.

[41] M. Busch, N. Koch, et al. NESSoS Deliverable D2.1 – First release of Method and Tool Evaluation, 2011. http://www.nessos-project.eu/media/deliverables/y1/NESSoS-D2.1.pdf.

[42] M. Busch, N. Koch, et al. NESSoS Deliverable D2.4 – Second Release of the Method and Tool Evaluation, 2013. http://www.nessos-project.eu/media/deliverables/y3/NESSoS-D2.4.pdf.

[43] M. Busch, N. Koch, M. Masi, R. Pugliese, and F. Tiezzi. Towards model-driven development of access control policies for web applications. In *Proceedings of the Workshop on Model-Driven Security*, MDsec '12, pages 41–46. ACM, 2012. doi:10.1145/2422498.2422502.

[44] M. Busch, N. Koch, and S. Suppan. Modeling Security Features of Web Applications. In Heisel et al. [100], pages 119–139. doi:10.1007/978-3-319-07452-8.

[45] M. Busch, N. Koch, and M. Wirsing. Evaluation of Engineering Approaches for Secure Software and Systems. In Heisel et al. [100], pages 234–265. doi:10.1007/978-3-319-07452-8.

[46] M. Busch, N. Koch, and M. Wirsing. SecEval: An Evaluation Framework for Engineering Secure Systems. In *Proceedings of Modellierung*, volume P-225 of *LNI*, pages 337–352, 2014. http://cs.emis.de/LNI/Proceedings/Proceedings225/337.pdf.

[47] M. Busch, M. Ochoa, and R. Schwienbacher. Modeling, Enforcing and Testing Secure Navigation Paths for Web Applications. Technical Report 1301, Ludwig-Maximilians-Universität München, 2013. http://uwe.pst.ifi.lmu.de/publications/TechReport1301.pdf.

[48] M. Busch and M. Wirsing. An Ontology for Secure Web Applications. In R. Lu, editor, *International Journal of Software and Informatics*, volume 9 of *International Journal of Software and Informatics*, pages 233–258. Institute of Software, Chinese Academy of Sciences, 2015.

[49] CBK. Common Body of Knowledge, 2015. http://nessos-project.eu/cbk.

[50] Center for Strategic and International Studies. Net losses: Estimating the global cost of cybercrime. economic impact of cybercrime ii. Technical report, Intel Security, 2014. http://www.mcafee.com/mx/resources/reports/rp-economic-impact-cybercrime2.pdf.

[51] Cenzic. Application vulnerability trends report. Technical report, Cenzic, 2014. https://www.trustwave.com/Resources/Library/Documents/Cenzic-Application-Vulnerability-Trends-2014/.

[52] Chandra, Pravir et al. Software assurance maturity model (samm) v. 1.0. http://www.opensamm.org/downloads/SAMM-1.0.pdf.

[53] M. R. Chaudron, W. Heijstek, and A. Nugroho. How effective is UML modeling? *Software & Systems Modeling*, 11(4):571–580, 2012. doi:10.1007/s10270-012-0278-4.

[54] Y. Cherdantseva and J. Hilton. A reference model of information assurance & security. In *Eighth International Conference on Availability, Reliability and Security (ARES)*, pages 546–555. IEEE, 2013.

[55] S. Chong, K. Vikram, and A. C. Myers. SIF: enforcing confidentiality and integrity in web applications. In *SS'07: Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, pages 1–16. USENIX Association, 2007. http://www.cs.cornell.edu/jif/sif/.

[56] S. A. Chun et al. iSecure Lab, 2014. http://cis.csi.cuny.edu/~project/iSecure/.

[57] CLUSIF Web application security working group. Web application security – managing web application security risks. Technical report, Club de la Sécurité de l'Information Français (CLUSIF), 2010. https://www.clusif.asso.fr/fr/production/ouvrages/pdf/CLUSIF-2010-Web-application-security.pdf.

[58] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton. The AETG system: An approach to testing based on combinatiorial design. *IEEE Trans. on Soft. Eng.*, 23(7):437–444, 1997.

[59] Common Criteria. Cc v3.1. release 4, 2012. https://www.commoncriteriaportal.org/cc/.

[60] Common Weakness Enumeration. CWE-840: Bussiness Logic Errors, 2015. http://cwe.mitre.org/data/definitions/840.html.

[61] J. Conallen. *Building Web applications with UML*. Addison-Wesley Longman Publishing Co., Inc., 2002.

[62] J. Cubo, J. Cuellar, S. Fries, J. A. Martín, F. Moyano, G. Fernández, M. C. F. Gago, A. Pasic, R. Román, R. T. Dieguez, and I. Vinagre. NESSoS Deliverable D11.2 – Selection and Documentation of the Two Major ApplicationCase Studies, 2011. http://www.nessos-project.eu/media/deliverables/y1/NESSoS-D11.2.pdf.

[63] J. Cuellar et al. NESSoS Deliverable D11.3 – Initial version of two case studies, evaluating methodologies, 2012. http://www.nessos-project.eu/media/deliverables/y2/NESSoS-D11.3.pdf.

[64] J. Cuellar et al. NESSoS Deliverable D11.4 – Pilot applications, evaluating NESSoS solutions, 2014.

[65] J. Cuellar and S. Suppan. A smart metering scenario, 2013. https://securitylab.disi.unitn.it/lib/exe/fetch.php?media=research_activities:erise:erise_2013:erise2013-smartmeteering-description.pdf.

[66] CWE. Common weakness enumeration, 2015. http://cwe.mitre.org/.

[67] J. Dehlinger and N. Subramanian. Architecting secure software systems using an aspect-oriented approach: A survey of current research. Technical report, Iowa State University, 2006.

[68] G. Denker, L. Kagal, and T. Finin. Security in the semantic web using {OWL}. *Information Security*, 10(1):51–58, 2005. doi:10.1016/j.istr.2004.11.002.

[69] G. Denker, L. Kagal, T. Finin, M. Paolucci, and K. Sycara. Security for daml web services: Annotation and matchmaking. In D. Fensel, K. Sycara, and J. Mylopoulos, editors, *The Semantic Web - ISWC 2003*, volume 2870 of *LNCS*, pages 335–350. Springer, 2003. doi:10.1007/978-3-540-39718-2_22.

[70] W. Diffie, P. C. Van Oorschot, and M. J. Wiener. Authentication and authenticated key exchanges. *Designs, codes and cryptography*, 2(2):107–125, 1992.

[71] D. Distante, P. Pedone, G. Rossi, and G. Canfora. Model-driven development of web applications with uwa, MVC and javaserver faces. In L. Baresi, P. Fraternali, and G.-J. Houben, editors, *Web Engineering*, volume 4607 of *Lecture Notes in Computer Science*, pages 457–472. Springer, 2007. doi:10.1007/978-3-540-73597-7_38.

[72] D. Dolev, C. Dwork, and M. Naor. Nonmalleable cryptography. *SIAM Review*, 45(4):727–784, 2003. doi:10.1137/S0036144503429856.

[73] Eclipse Foundation. XPand, 2013. http://wiki.eclipse.org/Xpand.

[74] Eclipse Foundation. Eclipse - An Open Development Platform, 2015. http://www.eclipse.org/.

[75] Eclipse Foundation. Eclipse Modeling Project, 2015. http://eclipse.org/modeling/.

[76] S. Eicker, M. Heisel, H. Schmidt, and W. Schwittek. NESSoS Deliverable D5.1 – Common Body of Knowledge, 2011. http://www.nessos-project.eu/media/deliverables/y1/NESSoS-D5.1.pdf.

[77] M. El-Attar. From misuse cases to mal-activity diagrams: bridging the gap between functional security analysis and design. *Software & Systems Modeling*, 13(1):173–190, 2014. doi:10.1007/s10270-012-0240-5.

[78] G. Elahi, E. Yu, and N. Zannone. A vulnerability-centric requirements engineering framework: analyzing security attacks, countermeasures, and requirements based on vulnerabilities. *Requirements Engineering*, 15(1):41–62, 2010. doi:10.1007/s00766-009-0090-z.

[79] A. Elçi. Keynote talk, sin'14: Standard ontology of security of information and networks?, 2014. http://www.sinconf.org/sin2014/images/SIN14_KeynoteTalk-AElci.pdf.

[80] D. Feledi, S. Fenz, and L. Lechner. Toward web-based information security knowledge sharing. *Information Security Technical Report*, 17(4):199–209, 2013. Special Issue: ARES 2012 7th International Conference on Availability, Reliability and Security. doi:10.1016/j.istr.2013.03.004.

[81] V. Felmetsger, L. Cavedon, C. Kruegel, and G. Vigna. Toward automated detection of logic vulnerabilities in web applications. In *Proceedings of the 19th USENIX conference on Security*, USENIX Security'10, page 10. USENIX Association, 2010. http://dl.acm.org/citation.cfm?id=1929820.1929834.

[82] D. Fensel. *Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce.* Springer, 2nd ed. 2004 edition, 2003.

[83] S. Fenz and A. Ekelhart. Formalizing information security knowledge. In *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*, ASIACCS '09, pages 183–194. ACM, 2009. http://securityontology.securityresearch.at/, doi:10.1145/1533057.1533084.

[84] K. Fisler, S. Krishnamurthi, L. A. Meyerovich, and M. C. Tschantz. Verification and change-impact analysis of access-control policies. In *ICSE*, pages 196–205. ACM, 2005.

[85] M. Fowler. *Domain-Specific Languages.* Addison-Wesley Professional, 1 edition, 2010. http://martinfowler.com/books/dsl.html.

[86] R. France, B. Rumpe, and M. Schindler. Why it is so hard to use models in software development: observations. *Software & Systems Modeling*, 12(4):665–668, 2013. doi:10.1007/s10270-013-0383-z.

[87] D. Fritsch. Modeling web security requirements in practice. Master's thesis, Ludwig-Maximilians-Universität München, 2015. Supervised by M. Busch.

[88] S. Gilmore, L. Gönczy, N. Koch, P. Mayer, M. Tribastone, and D. Varró. Non-functional Properties in the Model-Driven Development of Service-Oriented Systems. *J. Softw. Syst. Model.*, 10(3):287–311, 2011. doi:10.1007/s10270-010-0155-y.

[89] W. B. Glisson. *The Web Engineering Security (WES) methodology.* PhD thesis, University of Glasgow, 2008. http://theses.gla.ac.uk/186/.

[90] A. Gómez, M. Tellechea, and C. Rodríguez. D1.1 Requirements of AMI. Technical report, OPEN meter project, 2009. http://www.openmeter.com/files/deliverables/Open%20Meter_AMENDMENT%20to%20D11.pdf.

[91] GoPivotal Inc. Spring Security, 2013. http://static.springsource.org/.

[92] W. Gragido, D. Molina, J. Pirc, and N. Selby. *Blackhatonomics: An Inside Look at the Economics of Cybercrime.* Syngress, 2012.

[93] D. Groenewegen and E. Visser. Integration of data validation and user interface concerns in a DSL for web applications. In M. van den Brand, D. Gašević, and J. Gray, editors, *Software Language Engineering*, volume 5969 of *LNCS*, pages 164–173. Springer, 2010. doi:10.1007/978-3-642-12107-4_13.

[94] J. Grossman. Website security statistics report. Technical report, WhiteHat Security, 2013. https://www.whitehatsec.com/resource/stats.html.

[95] D. T. Group. Security requirement: Web applications. Technical Report 3.06, Deutsche Telekom AG, 2014. https://www.telekom.com/security.

[96] J. M. Guerrero. Microgrids: Integration of distributed energy resources into the smart-grid. In IEEE, editor, *IEEE International Symposium on Industrial Electronics*, pages 4281–4414, 2010.

[97] M. Hafner and R. Breu. *Security Engineering for Service-Oriented Architectures*. Springer, 2008.

[98] S. Hallé, T. Ettema, C. Bunch, and T. Bultan. Eliminating navigation errors in web applications via model checking and runtime enforcement of navigation state machines. In *Proceedings of the IEEE/ACM international conference on Automated software engineering*, ASE '10, pages 235–244. ACM, 2010. doi:10.1145/1858996.1859044.

[99] R. Hansen. Yahoo SEM Logic Flaw, 2008. http://projects.webappsec.org/w/page/13246943/Insufficient%20Process%20Validation.

[100] M. Heisel, W. Joosen, J. Lopez, and F. Martinelli, editors. *Advances in Engineering Secure Future Internet Services and Systems*. Number 8431 in LNCS State-of-the-Art-Surveys. Springer, 2014. doi:10.1007/978-3-319-07452-8.

[101] A. Herzog, N. Shahmehri, and C. Duma. An ontology of information security. In *International Journal of Information Security and Privacy*, pages 1–23, 2007. https://www.ida.liu.se/~iislab/projects/secont/, doi:jisp.2007100101.

[102] B. Hoisl and S. Sobernig. Integrity and confidentiality annotations for service interfaces in soaml models. In *Availability, Reliability and Security (ARES), 2011 Sixth International Conference on*, pages 673–679, 2011. doi:10.1109/ARES.2011.105.

[103] V. Igure and R. Williams. Taxonomies of attacks and vulnerabilities in computer systems. *Communications Surveys Tutorials, IEEE*, 10(1):6–19, 2008. doi:10.1109/COMST.2008.4483667.

[104] International Electrotechnical Commission (IEC). IEC 62351 Parts 1-8, Information Security for Power System Control Operations, 2007. https://www.smartgrid.gov/document/iec_62351_parts_1_8_information_security_power_system_control_operations.

[105] ISO/IEC. 27001: Information technology - Security techniques - Information security management systems - Requirements. Technical report, International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC), 2013.

[106] J. Hodges and C. Jackson and A. Barth. HTTP Strict Transport Security (HSTS). Specification, Internet Engineering Task Force (IETF), 2012. https://tools.ietf.org/html/rfc6797.

[107] M. Jakob. *Model-based engineering of web applications : the flashWeb method*. PhD thesis, Universität Stuttgart, 2011. http://elib.uni-stuttgart.de/opus/volltexte/2012/7023.

[108] JQuery Foundation. Jquery, 2015. https://jquery.com/.

[109] J. Jürjens. *Secure Systems Development with UML*. Springer, 2004. Tools and further information: http://www.umlsec.de/.

[110] J. Jürjens et al. Carisma UMLsec tools, 2014. http://carisma.umlsec.de.

[111] G. Kappel, B. Pröll, S. Reich, and W. Retschitzegger, editors. *Web Engineering: The Discipline of Systematic Development of Web Applications*. Wiley, 1 edition, 2006.

[112] A. Kim, J. Luo, and M. Kang. Security ontology to facilitate web service description and discovery. In S. Spaccapietra, P. Atzeni, F. Fages, M.-S. Hacid, M. Kifer, J. Mylopoulos, B. Pernici, P. Shvaiko, J. Trujillo, and I. Zaihrayeu, editors, *Journal on Data Semantics IX*, volume 4601 of *LNCS*, pages 167–195. Springer, 2007. doi:10.1007/978-3-540-74987-5_6.

[113] B. Kitchenham and S. Charters. Guidelines for performing Systematic Literature Reviews in Software Engineering. Technical Report EBSE 2007-001, Keele University and Durham University, 2007.

[114] M. Koch and F. Parisi-Presicce. UML specification of access control policies and their formal verification. *Software & Systems Modeling*, 5(4):429–447, 2006. doi:10.1007/s10270-006-0030-z.

[115] N. Koch. *Software Engineering for Adaptive Hypermedia Systems: Reference Model, Modeling Techniques and Development Process*. PhD thesis, Ludwig-Maximilians-Universität München, 2000. http://www.pst.informatik.uni-muenchen.de/personen/kochn/PhDThesisNoraKoch.pdf.

[116] N. Koch, A. Knapp, G. Zhang, and H. Baumeister. UML-Based Web Engineering - An Approach Based on Standards. In *Web Engineering: Modelling and Implementing Web Applications*, Human-Computer Interaction Series, pages 157–191. Springer, 2008.

[117] N. Koch and S. Kozuruba. Requirements models as first class entities in model-driven web engineering. In M. Grossniklaus and M. Wimmer, editors, *Current Trends in Web Engineering*, volume 7703 of *Lecture Notes in Computer Science*, pages 158–169. Springer, 2012. doi:10.1007/978-3-642-35623-0_16.

[118] V. Kolovski, J. A. Hendler, and B. Parsia. Analyzing Web Access Control Policies. In *WWW*, pages 677–686. ACM, 2007.

[119] S. Kozuruba. Modellbasierte Anforderungsanalyse für die Entwicklung von adaptiven RIAs. Master's thesis, Ludwig-Maximilians-Universität München, 2010. http://uwe.pst.ifi.lmu.de/publications/KozurubaDA.pdf.

[120] M. Krötzsch et al. Semantic mediawiki, 2015. https://semantic-mediawiki.org.

[121] M. Krötzsch et al. Semantic web, 2015. http://semanticweb.org.

[122] C. Lacek. In-depth comparison and integration of tools for testing security features of web applications, 2013. Bachelor thesis at Ludwig-Maximilians-Universität München, supervised by M. Busch.

[123] U. Lang et al. Privacyontology, 2015. http://www.privacyontology.org/.

[124] S. Lipner and M. Howard. The Trustworthy Computing Security Development Lifecycle. *Developer Network - Microsoft*, 2005. https://www.microsoft.com/en-us/sdl/. https://msdn.microsoft.com/en-us/library/ms995349.aspx.

[125] T. Lodderstedt, D. Basin, and J. Doser. SecureUML: A UML-Based Modeling Language for Model-Driven Security. In *Proc. 5th Int. Conf. Unified Modeling Language (UML'02)*, volume 2460 of *LNCS*, pages 426–441. Springer, 2002.

[126] F. Long, D. Mohindra, R. C. Seacord, D. F. Sutherland, and D. Svoboda. *The CERT Oracle Secure Coding Standard for Java (SEI Series in Software Engineering)*. Addison-Wesley Professional, 1 edition, 2011.

[127] M. S. Lund, B. Solhaug, and K. Stølen. *Model-Driven Risk Analysis – The CORAS Approach*. Springer, 2011.

[128] E. Martin. Automated test generation for access control policies. In *Companion to the 21st ACM SIGPLAN Symposium on Object-oriented Programming Systems, Languages, and Applications*, OOPSLA '06, pages 752–753. ACM, 2006. doi:10.1145/1176617.1176708.

[129] E. Martin and T. Xie. Automated Test Generation for Access Control Policies via Change-Impact Analysis. In *Proc. of Third International Workshop on Software Engineering for Secure Systems (SESS)*, pages 5–12, 2007.

[130] M. Masi, R. Pugliese, and F. Tiezzi. Formalisation and Implementation of the XACML Access Control Mechanism. In *ESSoS*, volume 7159 of *LNCS*, pages 60–74. Springer, 2012.

[131] F. P. McKenna. It won't happen to me: Unrealistic optimism or illusion of control? *British Journal of Psychology*, 84:39–39, 1993.

[132] S. Meliá, J. Gómez, S. Pérez, and O. Díaz. A Model-Driven Development for GWT-Based Rich Internet Applications with OOH4RIA. In *Proc. 8th Int. Conf. Web Engineering (ICWE'08)*, pages 13–23. IEEE, 2008.

[133] M. Menzel and C. Meinel. A Security Meta-model for Service-Oriented Architectures. In *Proc. 2009 IEEE Int. Conf. Services Computing (SCC'09)*, pages 251–259. IEEE, 2009.

[134] C. Miller and C. Valasek. Remote exploitation of an unaltered passenger vehicle. *Black Hat USA*, 2015. http://illmatics.com/Remote%20Car%20Hacking.pdf.

[135] P. Mohagheghi, W. Gilani, A. Stefanescu, M. Fernandez, B. Nordmoen, and M. Fritzsche. Where does model-driven engineering help? experiences from three industrial cases. *Software & Systems Modeling*, 12(3):619–639, 2013. doi:10.1007/s10270-011-0219-7.

[136] D. L. Moody. The method evaluation model: a theoretical model for validating information systems design methods. In *ECIS*, pages 1327–1336, 2003.

[137] T. Mouelhi, F. Fleurey, B. Baudry, and Y. Le Traon. A Model-Based Framework for Security Policy Specification, Deployment and Testing. In K. Czarnecki, I. Ober, J.-M. Bruel, A. Uhl, and M. Völter, editors, *Model Driven Engineering Languages and Systems*, volume 5301 of *LNCS*, pages 537–552. Springer, 2008. doi:10.1007/978-3-540-87875-9_38.

[138] F. Moyano, C. Fernandez-Gago, and J. Lopez. A conceptual framework for trust models. In S. Fischer-Hübner, S. Katsikas, and G. Quirchmayr, editors, *9th International Conference on Trust, Privacy & Security in Digital Business (TrustBus 2012)*, volume 7449 of *LNCS*, pages 93–104. Springer, Springer, 2012. doi:10.1007/978-3-642-32287-7.

[139] G. Mussbacher, D. Amyot, R. Breu, J.-M. Bruel, B. Cheng, P. Collet, B. Combemale, R. France, R. Heldal, J. Hill, J. Kienzle, M. Schöttle, F. Steimann, D. Stikkolorum, and J. Whittle. The relevance of model-driven engineering thirty years from now. In J. Dingel, W. Schulte, I. Ramos, S. Abrahão, and E. Insfran, editors, *Model-Driven Engineering Languages and Systems*, volume 8767 of *LNCS*, pages 183–200. Springer, 2014. doi:10.1007/978-3-319-11653-2_12.

[140] F. Nabi. Designing a secure framework method for secure business application logic integrity in e-commerce systems. *I. J. Network Security*, 12(1):29–41, 2011.

[141] National Energy Technology Laboratory. A vision for the smart grid. Report, 2009. http://www.netl.doe.gov/moderngrid/.

[142] H. Nergaard, N. Ulltveit-Moe, and T. Gjøsæter. A scratch-based graphical policy editor for xacml. In *ICISSP 2015 Proceedings of the 1st International Conference on Information Systems Security and Privacy ESEO, Angers, Loire Valley, France*, pages 182–191, 2015.

[143] NESSoS Project. Network of Excellence on Engineering Secure Future Internet Software Services and Systems, 2014. http://www.nessos-project.eu/.

[144] F. Neuhaus, A. Vizedom, K. Baclawski, M. Bennett, M. Dean, M. Denny, M. Grüninger, A. Hashemi, T. Longstreth, L. Obrst, et al. Towards ontology evaluation across the life cycle. In *Applied Ontology*, volume 8, pages 179–194. IOS Press, 2013. http://ontolog.cim3.net/cgi-bin/wiki.pl?OntologySummit2013_Communique, doi:10.3233/AO-130125.

[145] P. H. Nguyen, M. Kramer, J. Klein, and Y. L. Traon. An extensive systematic review on model-driven development of secure systems. *arXiv preprint arXiv:1505.06557*, 2015. http://arxiv.org/pdf/1505.06557.pdf.

[146] P. H. Nguyen, G. Nain, J. Klein, T. Mouelhi, and Y. Le Traon. Modularity and dynamic adaptation of flexibly secure systems: Model-driven adaptive delegation in access control management. In S. Chiba, r. Tanter, E. Bodden, S. Maoz, and J. Kienzle, editors, *Transactions on Aspect-Oriented Software Development XI*, volume 8400 of *LNCS*, pages 109–144. Springer, 2014. doi:10.1007/978-3-642-55099-7_4.

[147] No Magic Inc. Magicdraw, 2015. http://www.magicdraw.com/.

[148] N. F. Noy and D. L. McGuinness. Ontology development 101: A guide to creating your first ontology. Technical Report KSL-01-05, Stanford, 2011. http://www-ksl.stanford.edu/people/dlm/papers/ontology-tutorial-noy-mcguinness-abstract.html.

[149] OASIS. eXtensible Access Control Markup Language (XACML) Version 2.0, 2005. http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf.

[150] Object Management Group. XMI 2.1. Specification, OMG, 2005. http://www.omg.org/spec/XMI/.

[151] Object Management Group. Object Constraint Language (OCL) v2.3.1. Specification, OMG, 2012. http://www.omg.org/spec/OCL/2.3.1/.

[152] Object Management Group. Interaction Flow Modeling Language (IFML), FTF – Beta 2. Technical report, OMG, 2014. http://www.omg.org/spec/IFML/.

[153] L. Obrst, P. Chase, and R. Markeloff. Developing an ontology of the cyber security domain. In *STIDS*, pages 49–56, 2012. http://ceur-ws.org/Vol-966/STIDS2012_T06_ObrstEtAl_CyberOntology.pdf.

[154] M. Ochoa, J. Jürjens, and J. Cuéllar. Non-interference on UML state-charts. In *Objects, Models, Components, Patterns*, pages 219–235. Springer, 2012.

[155] M. Odersky et al. Scala, 2015. http://scala-lang.org/.

[156] L. Olsina, F. Papa, and H. Molina. How to measure and evaluate web applications in a consistent way. In *Web Engineering: Modelling and Implementing Web Applications*, pages 385–420. Springer, 2008.

[157] Open-Source Community. ownCloud, 2015. https://owncloud.org/.

[158] OpenLiberty. Easy XACML syntax with OpenAzPolicyReader, 2010. http://lists.openliberty.org/pipermail/openaz/2010-July/000074.html.

[159] Oracle. Java, 2015. http://www.java.com/.

[160] Oracle. MySQL, 2015. http://www.mysql.com/.

[161] OWASP Foundation. OWASP Top 10 – 2013, 2013. http://owasptop10.googlecode.com/files/OWASPTop10-2013.pdf.

[162] OWASP Foundation. Application Security Verification Standard (ASVS), 2014. https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project.

[163] OWASP Foundation. Clickjacking, 2014. https://www.owasp.org/index.php/Clickjacking.

[164] OWASP Foundation. Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet, 2014. https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet.

[165] OWASP Foundation. List of useful HTTP headers, 2014. https://www.owasp.org/index.php/List_of_useful_HTTP_headers.

[166] OWASP Foundation. OWASP Risk Rating Methodology, 2014. https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology.

[167] OWASP Foundation. OWASP Data Validation, 2015. https://www.owasp.org/index.php/Data_Validation.

[168] OWASP Foundation. Session fixation, 2015. https://www.owasp.org/index.php/Session_fixation.

[169] OWASP Foundation. SQL Injection Prevention Cheat Sheet, 2015. https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet.

[170] OWASP Foundation. User Privacy Protection Cheat Sheet, 2015. https://www.owasp.org/index.php/User_Privacy_Protection_Cheat_Sheet.

[171] J. Pauli. *The Basics of Web Hacking: Tools and Techniques to Attack the Web*. Syngress, 1 edition, 2013.

[172] C. P. Pfleeger and S. L. Pfleeger. *Security in Computing, 4th Edition*. Prentice Hall, 4th edition, 2006.

[173] PHP. Scripting Language, 2015. http://www.php.net/.

[174] Privacy Rights Clearinghouse. Chronology of data breaches, 2015. http://www.privacyrights.org/data-breach.

[175] G. Reggio, E. Astesiano, and C. Choppy. A framework for defining and comparing modelling methods. In R. De Nicola and R. Hennicker, editors, *Software, Services, and Systems*, volume 8950 of *Lecture Notes in Computer Science*, pages 377–408. Springer, 2015. doi:10.1007/978-3-319-15545-6_23.

[176] M. Reithmayer. Tool support for a knowledge base for secure software engineering. Master's thesis, Ludwig-Maximilians-Universität München, 2014. Supervised by M. Busch.

[177] A. Rodríguez, E. Fernández-Medina, and M. Piattini. Towards a UML 2.0 extension for the modeling of security requirements in business processes. In S. Fischer-Hübner, S. Furnell, and C. Lambrinoudakis, editors, *Trust and Privacy in Digital Business*, volume 4083 of *Lecture Notes in Computer Science*, pages 51–61. Springer, 2006. doi:10.1007/11824633_6.

[178] A. Rodríguez, E. Fernández-Medina, and M. Piattini. A BPMN extension for the modeling of security requirements in business processes. *IEICE transactions on information and systems*, 90(4):745–752, 2007.

[179] RWTH Aachen University. SWRL: A Semantic Web Rule Language Combining OWL and RuleML, 2004. http://www.w3.org/Submission/SWRL/.

[180] RWTH Aachen University. i* notation, 2014. http://istar.rwth-aachen.de.

[181] K. Rzehaczek. Transformation of graphical UWE models to a textual DSL, 2013. Bachelor thesis at Ludwig-Maximilians-Universität München, supervised by M. Busch.

[182] P. Salini and S. Kanmani. Ontology-based representation of reusable security requirements for developing secure web applications. *Int. J. Internet Technol. Secur. Syst.*, 5(1):63–83, 2013. doi:10.1504/IJITST.2013.058295.

[183] S. Schefer-Wenzl and M. Strembeck. Modellierungsunterstützung für die rollenbasierte delegation in prozessgestützten informationssystemen. *Wirtschaftsinformatik*, 56(4):237–260, 2014. `doi:10.1007/s11576-014-0433-3`.

[184] R. Schneiderman. Smart grid represents a potentially huge market for the electronics industry. *IEEE Signal Processing Magazine*, 27(5):8–15, 2010.

[185] B. Schneier. Attack trees. *Dr. Dobb's journal*, 24(12):21–29, 1999. https://www.schneier.com/paper-attacktrees-ddj-ft.html.

[186] B. Schneier. *Carry On: Sound Advice from Schneier on Security.* Wiley, 1 edition, 2013. https://www.schneier.com/essays/.

[187] T. Schreiber and A. Hoffmann. Sicherheit von webanwendungen maßnahmenkatalog und best practices. Technical Report 1, Bundesamt für Sicherheit in der Informationstechnik, 2006. https://www.bsi.bund.de/DE/Publikationen/Studien/websec/index_htm.html.

[188] S. Schreiner. Comparison of Security-related Tools and Methods for Testing Software, 2013. Bachelor thesis at Ludwig-Maximilians-Universität München, supervised by M. Busch.

[189] M. Schumacher. 6. toward a security core ontology. In *Security Engineering with Patterns*, volume 2754 of *LNCS*, pages 87–96. Springer, 2003. `doi:10.1007/978-3-540-45180-8_6`.

[190] R. Schwienbacher. Extending UWE with secure navigation paths, 2012. Bachelor thesis at Ludwig-Maximilians-Universität München, supervised by M. Busch.

[191] W. Schwinger, W. Retschitzegger, A. Schauerhuber, G. Kappel, M. Wimmer, B. Pröll, C. Cachero Castro, S. Casteleyn, O. De Troyer, P. Fraternali, I. Garrigos, F. Garzotto, A. Ginige, G.-J. Houben, N. Koch, N. Moreno, O. Pastor, P. Paolini, V. Pelechano Ferragud, G. Rossi, D. Schwabe, M. Tisi, A. Vallecillo, K. van der Sluijs, and G. Zhang. A survey on web modeling approaches for ubiquitous web applications. *International Journal of Web Information Systems*, 4(3):234–305, 2008. `doi:10.1108/17440080810901089`.

[192] D. Scott and R. Sharp. Abstracting application-level web security. In *Proceedings of the 11th international conference on World Wide Web*, WWW '02, pages 396–407. ACM, 2002. `doi:10.1145/511446.511498`.

[193] SDE. Service Development Environment, 2014. http://sde.pst.ifi.lmu.de/.

[194] Sensoria Project. Software Engineering for Service-Oriented Overlay Computers, 2011. http://www.sensoria-ist.eu/.

[195] M. Shema. *Hacking Web Apps: Detecting and Preventing Web Application Security Problems.* Syngress, 1 edition, 2012.

[196] J. Sherwood, A. Clark, and D. Lynas. *Enterprise Security Architecture: A Business-Driven Approach.* CRC Press, 1 edition, 2005.

[197] A. Shostack. *Threat Modeling: Designing for Security.* Wiley, 1 edition, 2014.

[198] G. Sindre and A. Opdahl. Eliciting security requirements with misuse cases. *Requirements Engineering*, 10(1):34–44, 2005.

[199] N. Slimani, H. Khambhammettu, K. Adi, and L. Logrippo. UACML: Unified Access Control Modeling Language. In *NTMS 2011*, pages 1–8, 2011.

[200] I. Sommerville. *Software Engineering (9th Edition)*. Pearson, 9 edition, 2010.

[201] A. Souag, C. Salinesi, I. Wattiau, and H. Mouratidis. Using security and domain ontologies for security requirements analysis. In *Computer Software and Applications Conference Workshops (COMPSACW), 2013 IEEE 37th Annual*, pages 101–107, 2013. doi:10.1109/COMPSACW.2013.124.

[202] A. Souag, C. Salinesi, and I. Comyn-Wattiau. Ontologies for security requirements: A literature survey and classification. In M. Bajec and J. Eder, editors, *Advanced Information Systems Engineering Workshops*, volume 112 of *Lecture Notes in Business Information Processing*, pages 61–69. Springer, 2012. doi:10.1007/978-3-642-31069-0_5.

[203] A. Souag, C. Salinesi, R. Mazo, and I. Comyn-Wattiau. A security ontology for security requirements elicitation. In *Engineering Secure Software and Systems*, pages 157–177. Springer, 2015.

[204] R. A. Spears. *Common American Phrases in Everyday Contexts: A Detailed Guide to Real-Life Conversation and Small Talk*. National Textbook Company, 1992.

[205] C. Sprenger et al. NESSoS Deliverable D9.4 – Enhanced Set of Solutions for Security Assurance for Services, 2013. http://www.nessos-project.eu/media/deliverables/y3/NESSoS-D9.4.pdf.

[206] Stanford Center for Biomedical Informatics Research. Webprotégé, 2015. http://protege.stanford.edu/products.php#web-protege.

[207] B. Stepien, A. P. Felty, and S. Matwin. A Non-technical User-Oriented Display Notation for XACML Conditions. In *MCETECH*, LNBIP 26, pages 53–64. Springer, 2009.

[208] R. Studer and S. Staab, editors. *Handbook on ontologies*. Springer, 2 edition, 2004.

[209] Sun Microsystems. Sun's XACML Implementation, 2006. http://sunxacml.sourceforge.net/.

[210] Symantec. Internet security threat report. Technical report, Symantec, 2015. http://www.symantec.com/de/de/security_response/publications/threatreport.jsp.

[211] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol. Version 1.2. Specification, Internet Engineering Task Force (IETF), 2008. http://tools.ietf.org/html/rfc5246.

[212] The MITRE Corporation. Common attack pattern enumeration and classification. mechanisms of attack., 2015. https://capec.mitre.org/data/definitions/1000.html.

[213] The MITRE Corporation. Common Vulnerabilities and Exposures List (CVE), 2015. https://cve.mitre.org.

[214] The MITRE Corporation. Structured threat information expression (stix), 2015. https://stix.mitre.org/.

[215] J. Tiago. Angriffsrisiken minimieren mit Security-Headern (German). *Heise IX Kompakt Security*, (4/2014):64–53, 2014.

[216] Transaction Processing Performance Council (TPC). TPC-W Benchmark, 2005. http://tpc.org/tpcw/.

[217] P. Valderas and V. Pelechano. A survey of requirements specification in model-driven development of web applications. *ACM Trans. Web*, 5(2):10:1–10:51, 2011. doi:10.1145/1961659.1961664.

[218] F. Valverde and O. Pastor. Applying Interaction Patterns: Towards a Model-Driven Approach for Rich Internet Applications Development. In *Proc. 7th Int. Wsh. Web-Oriented Software Technologies (IWWOST'08)*, 2008.

[219] A. van den Berghe, R. Scandariato, K. Yskout, and W. Joosen. Design notations for secure software: a systematic literature review. *Software & Systems Modeling*, pages 1–23, 2015. https://people.cs.kuleuven.be/~alexander.vandenberghe/review/overview.html, doi:10.1007/s10270-015-0486-9.

[220] H. van Tilborg and S. Jajodia, editors. *Encyclopedia of Cryptography and Security*. Springer, 2011. doi:10.1007/978-1-4419-5906-5.

[221] W3C Last Call Working Draft. Content Security Policy Level 2, 2014. http://www.w3.org/TR/CSP2/.

[222] A. Wali, S. A. Chun, and J. Geller. A bootstrapping approach for developing a cyber-security ontology using textbook index terms. In *Availability, Reliability and Security (ARES), 2013 Eighth International Conference on*, pages 569–576, 2013. doi:10.1109/ARES.2013.75.

[223] J. A. Wang and M. Guo. Security data mining in an ontology for vulnerability management. In *Bioinformatics, Systems Biology and Intelligent Computing, 2009. IJCBS '09. International Joint Conference on*, pages 597–603, 2009. doi:10.1109/IJCBS.2009.13.

[224] Web Engineering Group at LMU. UWE Website, 2015. http://uwe.pst.ifi.lmu.de/.

[225] D. Weinberger. *Too big to know: Rethinking knowledge now that the facts aren't the facts, experts are everywhere, and the smartest person in the room is the room.* Basic Books, 2011.

[226] J. Weinberger, P. Saxena, D. Akhawe, M. Finifter, R. Shin, and D. Song. An empirical analysis of xss sanitization in web application frameworks. Technical report, University of California at Berkeley, 2011. http://www.eecs.berkeley.edu/Pubs/TechRpts/2011/EECS-2011-11.pdf.

[227] Wikimedia Foundation et al. Wikidata, 2015. https://www.wikidata.org.

[228] K. Wolf. Sicherheitsbezogene Model-to-Code Transformation für Webanwendungen (German). Master's thesis, Ludwig-Maximilians-Universität München, 2012. Diplomarbeit, supervised by M. Busch.

[229] E. Z. Yang et al. HTML purifier – standards-compliant HTML filtering, 2015. http://htmlpurifier.org/.

[230] G. Zhang, N. Koch, and A. Knapp. Aspect-Oriented Modeling of Access Control in Web Applications. In *6th Workshop on Aspect Oriented Modeling (AOM)*, 2005.

All online resources were lastly accessed in January 2016.