Modeling Security Features of Web Applications*

Marianne Busch[1], Nora Koch[1], and Santiago Suppan[2]

[1] Institute for Informatics, Ludwig-Maximilians-Universität München
Oettingenstraße 67, 80538 München, Germany
{busch, kochn}@pst.ifi.lmu.de
[2] Siemens AG, Germany
Otto-Hahn-Ring 6, 81739 München, Germany
santiago.suppan.ext@siemens.com

Abstract. Securing web applications is a difficult task not only, because it is hard to implement bulletproof techniques, but also because web developers struggle to get an overview of how to avoid security flaws in a concrete application. This is aggravated by the fact that the description of a web application's security concept is often scattered over lengthy requirements documents, if documented at all. In this chapter, we extend the graphical, UML-based Web Engineering (UWE) language to model security concepts within web applications, thus providing the aforementioned overview. Our approach is applied to a case study of an Energy Management System that provides a web interface for monitoring energy consumption and for configuring appliances. Additionally, we give an overview of how our approach contributes to the development of secure web applications along the software development life cycle.

Keywords: UML-based web engineering, secure web engineering, web applications, UML, security, Energy Management System, Smart Home

1 Introduction

The rising cybercrime as well as the growing awareness of data privacy due to global surveillance disclosures implies an urgent need to secure web applications. Besides confidential connections and authentication, both data access control and navigational access control are the most relevant security features in this field. However, adding such security features to already implemented web applications is an error-prone task.

Therefore, the goal is to include security features in early stages of the development process of web applications, i.e., at requirements specification and design modeling level. Secure web engineering approaches as ActionGUI [1] and the UML-based Web Engineering (UWE) [2] have been developed, trying to abstract from as many implementational details as possible.

* This work has been supported by the EU-NoE project NESSoS, GA 256980.

The way that seems right for most modeling methods, raises questions when dealing with design decisions related to more web-security specific concerns: How should Cross-Site-Request-Forgery (CSRF) [3] be prevented, in order to avoid end users to execute unintentionally malicious actions for an attacker, on a web application in which they are currently authenticated? What should happen in case the web application is under attack, e.g., under denial-of-service attack, making the machine or network resource unavailable to its intended users?

So far, those questions tend to be answered in lengthy specification documents or they are just documented by the code itself. However, a straightforward understanding of the way how web security is managed for a certain web application is crucial.

Our approach aims at addressing the answer to these questions at design level, extending the set of modeling elements provided by the UWE language in order to be able to express protection-specific security concerns. The challenge is to find means for recording security-related design decisions for the web, while maintaining the necessary abstraction a modeling language needs. Therefore, we extend UWE's UML profile to support modeling solutions that should be deployed to shield a web application and its users against attacks. Such is the aim of providing language elements to model features like *CSRF prevention* and *injection prevention* or special behavior for the case that an application is *under attack*.

In this chapter, we not only introduce the latest UWE extension along with a case study about the web interface of an Energy Management System (EMS), but also give an overview of how UWE's security features, which have been developed within the EU project NESSoS [4], support the phases of the software development life cycle (SDLC). This begins with the requirement and design phase, where UWE enables security engineers to get an overview of the application, but also serves as a notation for documentation. Aside from that, UWE models can be used as input for tools that generate artifacts for the implementation. For the testing phase, two approaches are available, (1) for testing that a user can only navigate the web application using a predefined path and (2) a toolchain for testing access control policies generated from UWE models. For the latter, the interested reader is referred to chapter [5].

The remainder of this chapter is structured as follows: In Sect. 2 we introduce the EMS case study, which is our running example. Section 3 gives an overview of security features that play a major role in the web and which are required for our case study, before the UWE approach is introduced in Sect. 4. In Sect. 5 we describe our UWE extensions for protection-specific security concerns, by applying them to the case study. Section 6 positions UWE in the SDLC. Finally, we present related work in Sect. 7 and conclude in Sect. 8.

2 Case Study: Energy Management System

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 in the next section into more security-related details.

2.1 Components of Smart Homes

Figure 1 visualizes the entities in a Smart Home. Generally, the EMS is an interface for the Smart Grid customer that visualizes consumption data. Concrete instantiations can be realized by means of mature web application technology, which provides several ways of advanced functionality, as 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.

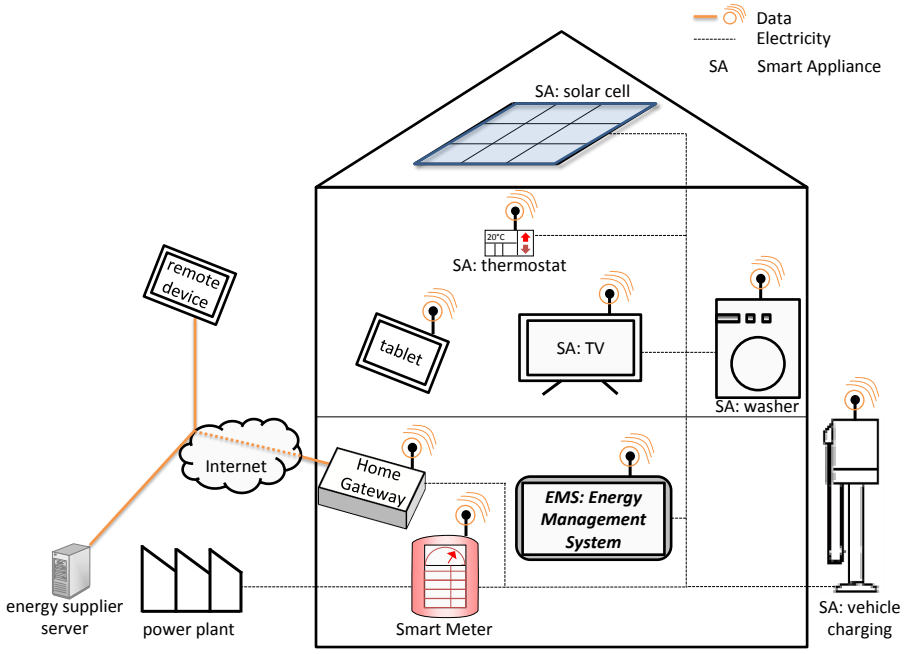

Fig. 1. Entities in the Smart Home (adapted from [6])

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.

A possibility to control energy consumption more globally is Demand Side Management. It envisions to adapt 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.

2.2 Actors

According to [7], 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[3] when the prices are higher. We also refer to the prosumer simply as "(private) user" or "customer".

Figure 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., underaged 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 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.

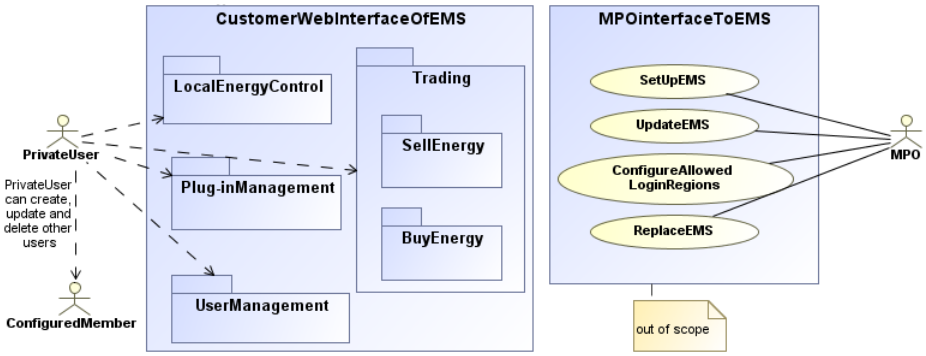

Fig. 2. Requirements overview (UML use case diagram)

[3] The term microgrid [8] refers to areas where small communities trade local energy, in addition to the energy supply provided by professional energy suppliers.

2.3 Functionality

The main functionality of the EMS is shown in Fig. 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 Fig. 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.

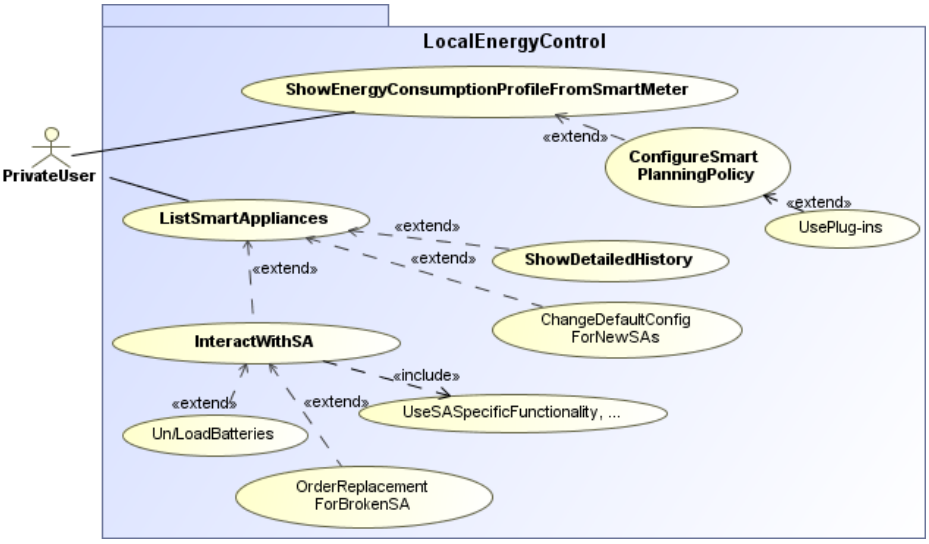

Fig. 3. Requirements of local energy control (UML use case diagram)

Easy access to real-time information supports the users, e.g., to pay attention to their energy consumption, as depicted at the top of Fig. 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.

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. 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?) functions as follows: The third party provides a plugin which 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, if the prosumer has defined a policy that allows the process to negotiate automatically. 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. 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.

Fundamentally, the customer will access the EMS as the central administration point. No process should demand direct interaction between the customer and an external third party service (provided as a website or otherwise). 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 the personal information a certain plugin can access and the functions a plugin can execute.

User Management. Prosumers can allow other persons to log into the web application. 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.

In this work, we focus on the EMS. The interested reader is referred to [9, example section / EMS] for all use case diagrams. Additionally, more comprehensive descriptions of general Smart Grid functional requirements can be found in [10,11].

3 Secure Web Applications

This section introduces common security features, including those which are special for web applications. Security features, detailed in the following, are: authentication, panic mode, reauthentication, secure connections, authorization, user zone concept, cross-site-request-forgery prevention, under attack mode and SQL-injection prevention.

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 users 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., HTTP Strict Transport Security (HSTS) 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 [12]. 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" [13]. 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.

As announced in the introduction, some security features can be handled at an abstract level, as e.g., authorization, whereas others are to be thought of at the end of the design phase, as SQL-injection prevention. Note that we do not claim to cover all possible web security features, although we try to cover the most common ones.

4 Overview of UML-based Web Engineering (UWE)

This section introduces the modeling language UML-based Web Engineering (UWE) [9,2], which we use to model secure web applications.

One of the cornerstones of the UWE language is the "separation of concerns" principle using separate models for the different views of a web application, such as the navigation and the presentation view. However, we can observe that security features are cross-cutting concerns which cannot be separated completely. The views and corresponding UWE models are:

Requirements View defines (security) requirements of a project.
Content View contains the data structure used by the application.
Access Control View is given by a *UWE Role Model* and a *Basic Rights Model*. The former describes the hierarchy of user groups to be used for authorization and access control issues. It is usually part of a *User Model*, which specifies basic structures, as e.g., that a user can take on certain roles simultaneously. The latter defines the access control policies. It constrains elements from the *Content Model* and from the *Role Model*.
Presentation View sketches the web application's user interface.
Process View details the flow of actions to be executed.
Navigation View defines the navigation flow of the application and navigational access control policies. The former shows which possibilities of navigation exist in a certain context. The latter specifies which roles are allowed to navigate to a specific state and the action taken in case access cannot be granted. 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. Furthermore, secure connections between server and browser are modeled, too.

The following table maps UWE views to security features that they can express. We introduce concrete modeling elements for this security features in the next section.

View	**Security Features**
Content	SQL-injection prevention, cross-site-request-forgery prevention
Navigation	authentication, reauthentication, secure connections, under attack mode
Access Control	authorization, under attack mode, user zone concept
Process	user zone concept, panic mode

For each view, an appropriate type of UML diagram is selected, e.g., a state machine for the navigation model. The UWE approach defines a UML profile that consists of a set of stereotypes, tag definitions, constraints and patterns for modeling secure web applications. The profile can be downloaded from the UWE website [9].

Stereotypes can be applied to UML model elements (e.g. classes, states, dependencies) and values can be assigned to tags. UML tags are always associated to a stereotype and stereotypes can inherit tags from other stereotypes. In the UWE profile, patterns are provided for modeling widely used elements, as e.g., different types of authentication mechanisms.

5 Designing Secure Web Applications with UWE

This section shows how to model security features with the most recently introduced UWE profile elements. The main advantages of these specific modeling elements for the modeler are on the one hand to promote the inclusion of security aspects from the early phases of the development. On the other hand it enables documentation and, due to brevity, a quick understanding of security features that are or should be employed. The elements are introduced using our EMS case study.

5.1 Content View

When modeling larger web 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 Fig. 4. 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. Fig. 2): local energy control, user management, energy trade system and plugin management. As can be seen in Fig. 4, 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 planing or trading plugin.

As far as security is concerned, the UWE profile redefines the UML stereotype ≪component≫ with the following tags:

csrfPrevention models how cross-site-request-forgery (CSRF) should be repelled. The modeler can choose from the options presented at the OWASP

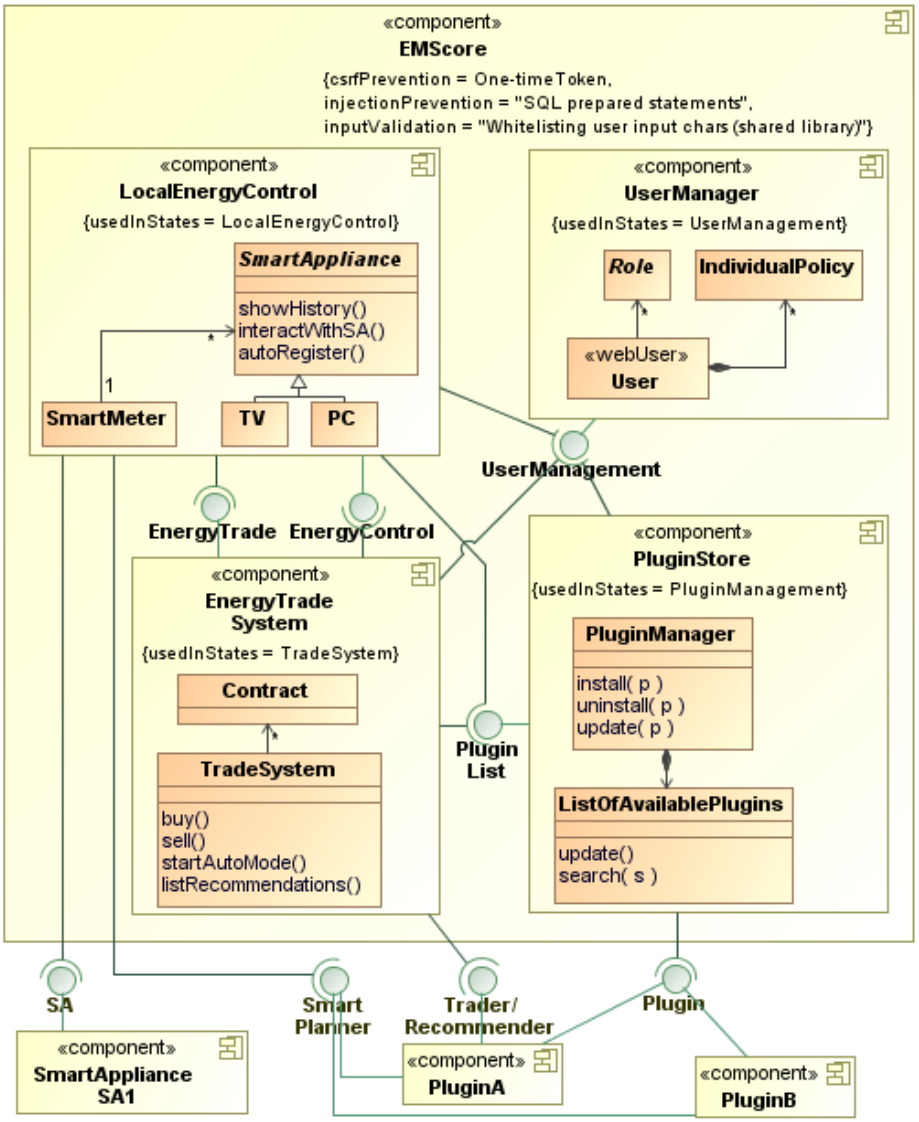

Fig. 4. UWE: Content model

CSRF Cheat Sheet[4]. 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.

injectionPrevention records how SQL injections (and others 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.

Additionally to these security features, the UWE profile provides the tag {usedInStates} to denote in which state of the application a certain component is used. More about states can be found in Sect. 5.3. 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.

5.2 Role and Access Control View

Figure 5 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, although this is not shown in the figure.

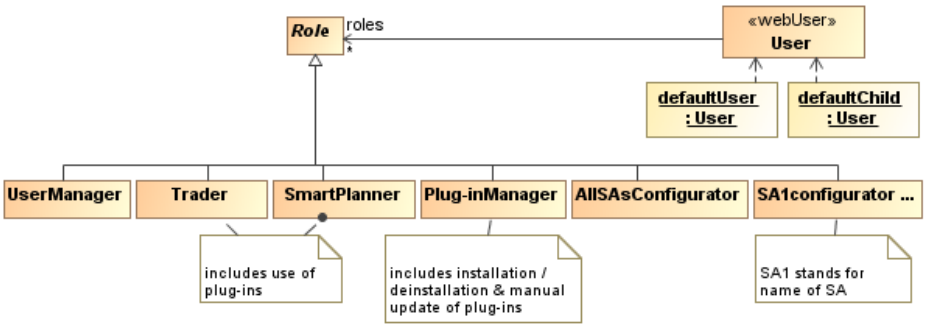

Fig. 5. UWE: Role model

Defining access control for web applications, has already been described in [2]. For our EMS application, Fig. 6 shows an excerpt. For example, someone with

[4] OWASP CSRF Prevention Cheat Sheet. `https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet`

the role `UserManager` is allowed to «delete» users, as long as the «authorization-Constraint», which has to be specified in OCL [14], 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.

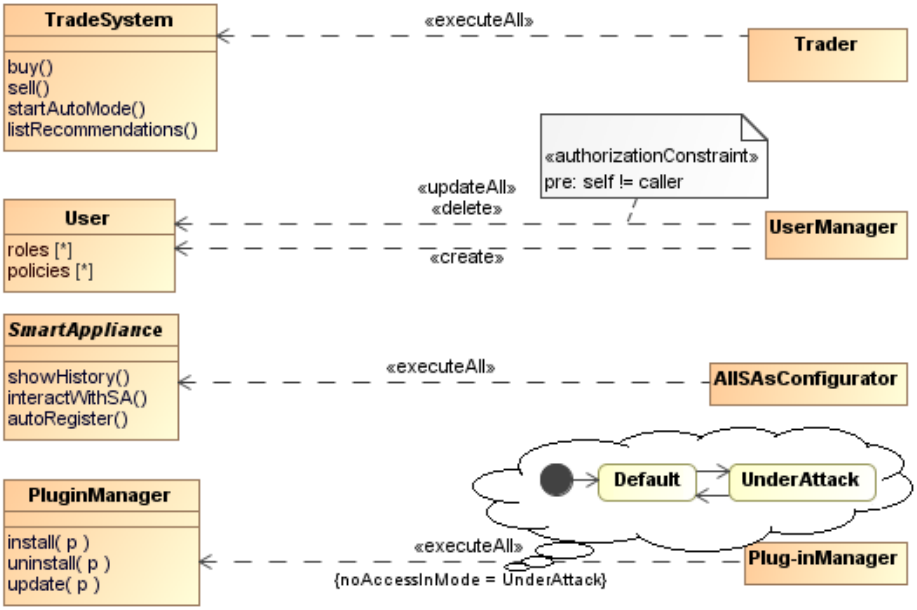

Fig. 6. UWE: Basic Rights model excerpt

Regarding our security requirements, the UWE Basic Rights model has to be extended to enable the specification of different modes. Therefore, the tag {noAccessInMode} is added. It allows to choose from a set of states in which the application 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 [15], the UWE profile with its typed tags makes sure that the value for the tag can only be chosen from all available state elements.

As depicted at the bottom of Fig. 6, the EMS only allows to install plugins when it is not under attack.

5.3 Navigation and Process View

User navigation is one of the most distinguished web features. Since 2011, UML state charts are used in UWE to express the navigation possibilities a user has within a certain state of the web application [2]. 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. Fig. 7).

The stereotype «integratedMenu» is defined to be a shortcut for showing menus entries for all menus of Submachine States, in case the user is allowed to access them. Submachine 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 Fig. 7, 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}.

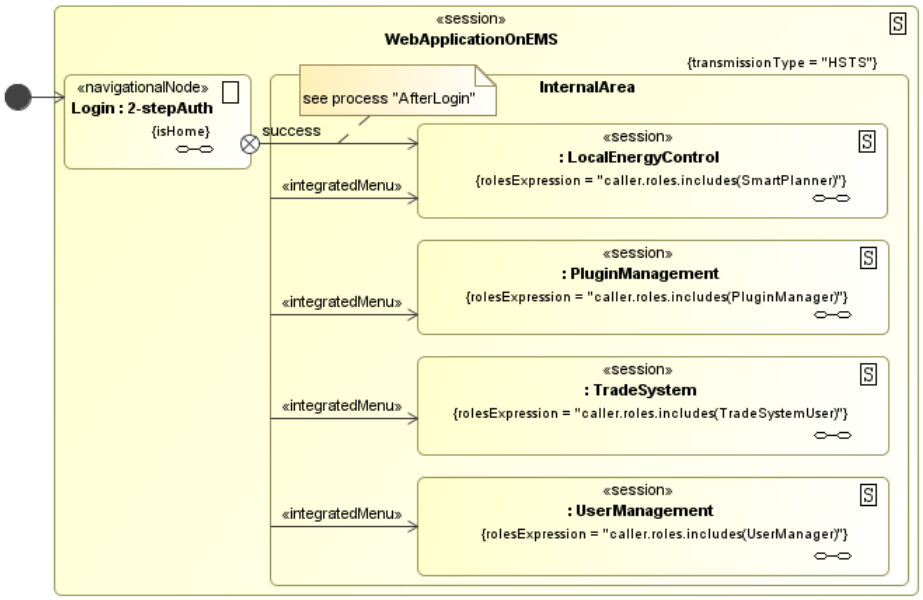

Fig. 7. UWE: Navigation model overview

If needed, activity diagrams can be added to detail the process that is executed behind the scenes. For example, Fig. 8 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.

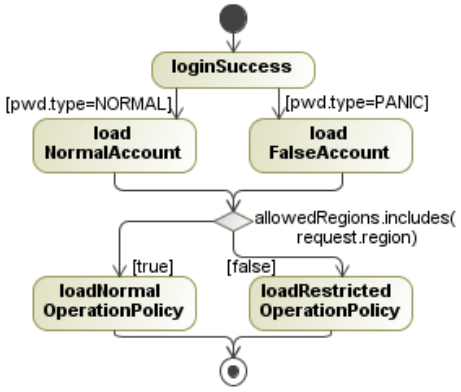

Fig. 8. UWE: Process after successful login

Exemplarily, the submachine state diagram of the plugin management is shown in Fig. 9. 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`.

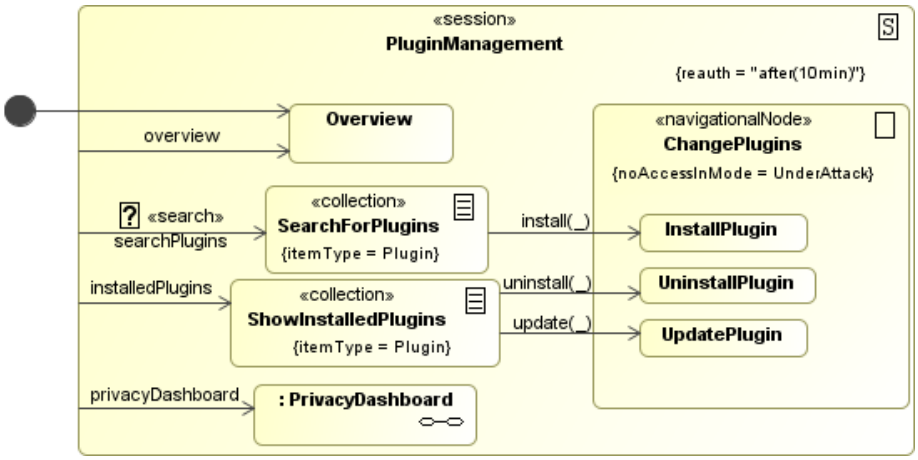

Fig. 9. UWE: Navigation model for plugin management

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 {noAccessInMode} 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.

All diagrams of our EMS case study can be found on the UWE web page [9, example section / EMS]; the original model can be downloaded as MagicDraw project or XMI file.

6 UWE in the Software Development Life Cycle

We consider an iterative Service Development Life Cycle (SDLC), consisting of at least the phases: requirements, design, implementation, testing and deployment [16]. In the following, we show how security-related UWE extensions developed during the NESSoS project [4] can be positioned in this development process.

In the *requirements phase*, use case diagrams and coarse-grained activity diagrams record customer wishes. UWE enhances requirements models using web-specific stereotypes, e.g., to denote use cases which require server-side processing. However, the main focus of UWE is on the *design phase*, in which the rest of the above-mentioned models are created or updated. To ease the design of web applications with UWE within the CASE tool MagicDraw [15], the plugin MagicUWE [17] has been developed. MagicUWE is presented in more detail in chapter [5].

One of the main advantages of models built using UWE is to get an overview of the web application and to quickly provide an impression of what is important in the different views on it. This is especially needed when new developers join an existing project, because clean documentation serves as a basis for a concise introduction that helps to avoid misunderstandings. Being clear about the conceptual structure of an application is of major importance when securing a web application, as a single misconception or thoughtlessness can lead to a vulnerability. If exploited, this vulnerability might then cause privacy violation for customers, reputation damage of a company or financial damage right up to bankruptcy or lawsuit. Providing an overview of a web application is also valuable for documentation and for discussions between modelers and software developers in order to reduce misunderstandings at the transition between design and implementation.

Constraints as "customers can only submit their order after they have selected a credit card to pay with" can be inferred from UWE Navigational State models. How to extract so called Secure Navigation Paths (SNPs) and how to use them for the generation of a monitor that shields the web application from illicit access sequences, is described in [18]. Prototypical tool support can be downloaded from [9].

Within the NESSoS project not only the web modeling approach UWE has been improved, but also ActionGUI has been developed further, which strives to implementing the whole application logic from models (cf. Sect. 7). Due to ActionGUI's different focus, it has been interesting to consider a model-to-model transformation from UWE to ActionGUI, as described in [19].

In the *implementation phase* code is written, based on the models. Tool support is preferable, but as can be expected, code generation is only possible where detailed information is given in the models. Therefore, modelers have to balance the need for abstraction against the need for detailed information. Consequently, UWE does not aim to generate complete web applications, as it turned out to overload models and the maintenance for a code generator like UWE2JSF [20] became unreasonable high, in order to keep up with the rapid development of web features. Nonetheless, it has proven to be helpful to transform some parts of the UWE models to code or other implementation-related artifacts. An example is the specification of role based access control (RBAC) rules. The model to text transformation language XPand [21] is used to transform UWE Basic Right models to XACML [22] and to code snippets. For the latter, a prototypic transformation of the data structure, roles and RBAC rules to Apache Wicket with Apache Shiro and Hibernate has been implemented [23].

Exporting XACML policies, which can include RBAC policies from the Basic Rights model as well as from the navigational states model, is implemented in a tool called UWE2XACML. In [24], Busch et al. explain how XACML can be transformed to FACPL [25], a formal policy language with the advantage of fully specified semantics. The transformation comprises several tools, which are integrated in the Service Development Environment (SDE) [26,16]. The SDE is a tool workbench, which allows to build tool chains of tools that are integrated, i.e. that provide a wrapper for the SDE.

As far as the *testing phase* of the SDLC is concerned, UWE's Basic Rights model can be the starting point for generating test cases by using a tool chain, as described in chapter [5]. The advantage is that policies are modeled at a high level of abstraction so they are easy to understand and to maintain, whereas policies written in XACML tend to become lengthy and error-prone so that thorough testing is mandatory.

In addition to enforcing Secure Navigation Paths by monitors (as introduced above), the modeled paths can also be used for automatic testing to check that a web application correctly prohibits attempts to break out of the predefined navigation structure [18].

7 Related Work

This section introduces related work for the Energy Management System (EMS) case study and approaches for secure web engineering.

Energy Management Systems. The EMS is a component of a Smart Grid. Unfortunately, literature [11,27,28] does not offer a coherent view of the components

of a Smart Grid. For our case study, we rely on the components as described in [6,29].

In [11], the Energy Management System is described as a *consumption display unit*, which is regarded as an optional device that helps advanced metering infrastructure (AMI) objectives, i.e., Demand Side Management events. For sustainable energy supply, Demand Side Management has to be considered as a key technology. Furthermore, the Energy Management System supports the user in interacting with the Smart Home, which is another key to the successful acceptance of the Smart Grid. This requires to manifest the EMS as a crucial part of the Smart Grid.

The Energy@Home Project[5] illustrates Smart Meter and Home Energy Management implementations. The overall description matches with our case study, but it does not give any insight on security or privacy.

The OpenNode Project [27] emphasizes research on electrical distribution grid operation. The prosumer endpoint is mentioned, but it does left out any details on the required end point functionality. From a holistic point of view, the OpenNode architecture is complementary with our view of Smart Homes and the proposed EMS functionality depicted in this chapter.

The British Department of Energy and Climate Change give in their technical reports in [28] detailed functional requirements on the Smart Home including technical and functional descriptions of the Energy Management System ("In Home Display") in the report. The report's functional requirements are equivalent to the functionality from our case study. Security requirements on the other hand are referenced, but clearly not in the report's scope.

Secure web application modeling. Existing approaches are briefly introduced in the following, adapted from [30].

ActionGUI [1] 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.

UMLsec [31] 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 adopted from UML1.4 to UML2. However, the new tools[6] have not been updated for almost two years.

SecureUML [32] is a UML-based modeling language for secure systems. It provides modeling elements for role-based access control and the specifica-

[5] Energy@Home. `http://www.enel.com/en-GB/innovation/smart_grids/smart_homes/smart_info/`

[6] UMLsec tools. `http://carisma.umlsec.de`

tion 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 our approach, we specify role-based execution rights to methods in a basic rights model using dependencies instead of the SecureUML association classes, which 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.

A similar approach is UACML [33] 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 overloaded, as SecureUML uses association classes instead of dependencies and UACML does not introduce a separate model to specify user-role hierarchies.

Other approaches address modeling of security aspects of service-oriented architectures (SOAs), such as the SECTET framework [34], UML4SOA [35], and SecureSOA [36]. The first one 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 the latter relies on FMC block diagrams and BPMN notation.

8 Conclusion and future work

In summary, it can be stated that our approach for engineering secure web applications using UWE contributes to the task of securing web applications. Consequently, a long-term impact should be the reduction of security flaws and of necessary security patches. As it is not easy to measure the long-term impact of UWE, we at least can tell that UWE helps to get clear about which security features are important for certain functions of concrete web applications. In particular, UWE addresses security features starting in the early phases of development.

For future work, we plan to include more web-specific security features and to validate our approach by modeling further case studies. Additionally, we extend our approach to cover model validation. Therefore, we are working on a textual version of UWE, called TextualUWE, which is based on a domain specific language. Our aim is to use functional Scala on TextualUWE to check for inconsistencies in the models, as unreachable navigational states or contradictory access control rules. Besides, it would also be interesting to investigate on transferring UWE's security concepts to other web modeling languages that have not yet incorporated security features.

References

1. Basin, D., Clavel, M., Egea, M.: Automatic Generation of Smart, Security-Aware GUI Models. In: Engineering Secure Software and Systems. Volume 5965 of LNCS., Springer (2010) 201–217

2. Busch, M., Knapp, A., Koch, N.: Modeling Secure Navigation in Web Information Systems. In Grabis, J., Kirikova, M., eds.: 10th International Conference on Business Perspectives in Informatics Research. LNBIP, Springer Verlag (2011) 239–253
3. Barth, A., Jackson, C., Mitchell, J.C.: Robust defenses for cross-site request forgery. In: Proceedings of the 15th ACM Conference on Computer and Communications Security. CCS '08, New York, NY, USA, ACM (2008) 75–88
4. NESSoS: Network of Excellence on Engineering Secure Future Internet Software Services and Systems. http://nessos-project.eu/ (2014)
5. Bertolino, A., Busch, M., Daoudagh, S., Lonetti, F., Marchetti, E.: A Toolchain for Designing and Testing Access Control Policies. In Lopez, J., Martinelli, F., eds.: Advances in Engineering Secure Future Internet Services and Systems. Volume LNCS 8431., Springer (2014)
6. Cuellar, J., Suppan, S.: A smart metering scenario (2013) https://securitylab.disi.unitn.it/lib/exe/fetch.php?media=research_activities:erise:erise_2013:erise2013-smartmeteering-description.pdf.
7. Cuellar, J.: NESSoS deliverable D11.4 – Pilot applications, evaluating NESSoS solutions. to appear (2014)
8. Guerrero, J.M.: Microgrids: Integration of distributed energy resources into the smart-grid. In: IEEE International Symposium on Industrial Electronics. (2010) 4281–4414
9. LMU. Web Engineering Group.: UWE Website. http://uwe.pst.ifi.lmu.de/ (2014)
10. Cubo, J., Cuellar, J., Fries, S., Martín, J.A., Moyano, F., Fernández, G., Gago, M.C.F., Pasic, A., Román, R., Dieguez, R.T., Vinagre, I.: Selection and documentation of the two major applicationcase studies. NESSoS deliverable D11.2 (2011)
11. Gómez, A., Tellechea, M., Rodríguez, C.: D1.1 Requirements of AMI. Technical report, OPEN meter project (2009)
12. Bennett, C., Wicker, S.: Decreased time delay and security enhancement recommendations for ami smart meter networks. In: Innovative Smart Grid Technologies (ISGT). (2010) 1–6
13. OWASP Foundation: OWASP Top 10 – 2013 (2013) http://owasptop10.googlecode.com/files/OWASPTop10-2013.pdf.
14. OMG.: OCL 2.0. http://www.omg.org/spec/OCL/2.0/ (2011)
15. No Magic Inc.: Magicdraw. http://www.magicdraw.com/ (2014)
16. Busch, M., Koch, N.: NESSoS Deliverable D2.3 – Second Release of the SDE for Security-Related Tools. (2012)
17. Busch, M., Koch, N.: MagicUWE — A CASE Tool Plugin for Modeling Web Applications. In: Proc. 9th Int. Conf. Web Engineering (ICWE'09). Volume 5648 of LNCS., Springer (2009) 505–508
18. Busch, M., Ochoa, M., Schwienbacher, R.: Modeling, Enforcing and Testing Secure Navigation Paths for Web Applications. Technical Report 1301, Ludwig-Maximilians-Universität München (2013)
19. Busch, M., García de Dios, M.A.: ActionUWE: Transformation of UWE to ActionGUI Models. Technical report, Ludwig-Maximilians-Universität München (2012) Number 1203.
20. Kroiß, C., Koch, N., Knapp, A.: UWE4JSF - A Model-Driven Generation Approach for Web Applications. In Gaedke, M., Grossniklaus, M., Díaz, O., eds.: Proc. 9th Int. Conf. Web Engineering (ICWE'09). LNCS 5648, Springer Berlin (2009) 493–496

21. Eclipse: XPand. http://wiki.eclipse.org/Xpand (2013)
22. OASIS: eXtensible Access Control Markup Language (XACML) Version 2.0. http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf (2005)
23. Wolf, K.: Sicherheitsbezogene Model-to-Code Transformation für Webanwendungen (German) (2012) Bachelor Thesis.
24. Busch, M., Koch, N., Masi, M., Pugliese, R., Tiezzi, F.: Towards model-driven development of access control policies for web applications. In: Model-Driven Security Workshop in conjunction with MoDELS 2012, ACM Digital Library (2012)
25. Masi, M., Pugliese, R., Tiezzi, F.: Formalisation and Implementation of the XACML Access Control Mechanism. In: ESSoS. LNCS 7159, Springer (2012) 60–74
26. SDE: Service Development Environment. http://www.nessos-project.eu/sde (2014)
27. Soriano, R., Alberto, M., Collazo, J., Gonzales, I., Kupzo, F., Moreno, L., Lugmaier, A., Lorenzo, J.: OpenNode. Open Architecture for Secondary Nodes of the Electricity SmartGrid. In: 21st International Conference on Electricity Distribution. (2011)
28. Department of Energy and Climate Change: Smart Metering Implementation Programme, Response to Prospectus Consultation, Overview Document. Technical report, Office of Gas and Electricity Markets (2011)
29. Beckers, K., Fabender, S., Heisel, M., Suppan, S.: A threat analysis methodology for smart home scenarios. In: SmartGridSec 14, Springer LNCS (2014)
30. Busch, M.: Secure Web Engineering supported by an Evaluation Framework. In: Modelsward 2014, Scitepress (2014)
31. Jürjens, J.: Secure Systems Development with UML. Springer (2004) Tools and further information: http://www.umlsec.de/.
32. Lodderstedt, T., Basin, D., Doser, J.: SecureUML: A UML-Based Modeling Language for Model-Driven Security. In: Proc. 5th Int. Conf. Unified Modeling Language (UML'02). Volume 2460 of LNCS., Springer (2002) 426–441
33. Slimani, N., Khambhammettu, H., Adi, K., Logrippo, L.: UACML: Unified Access Control Modeling Language. In: NTMS 2011. (2011) 1–8
34. Hafner, M., Breu, R.: Security Engineering for Service-Oriented Architectures. Springer (2008)
35. Gilmore, S., Gönczy, L., Koch, N., Mayer, P., Tribastone, M., Varró, D.: Non-functional Properties in the Model-Driven Development of Service-Oriented Systems. J. Softw. Syst. Model. **10**(3) (2011) 287–311
36. Menzel, M., Meinel, C.: A Security Meta-model for Service-Oriented Architectures. In: Proc. 2009 IEEE Int. Conf. Services Computing (SCC'09), IEEE (2009) 251–259