Modeling Secure Navigation in Web Information Systems*

Marianne Busch[1], Alexander Knapp[2], and Nora Koch[1,3]

[1] Ludwig-Maximilians-Universität München,
Oettingenstraße 67, 80538 München, Germany
{busch,kochn}@pst.ifi.lmu.de
[2] Universität Augsburg,
Universitätsstraße 6a, 86159 Augsburg, Germany
knapp@informatik.uni-augsburg.de
[3] Cirquent GmbH,
Arabellastraße 17, 81925 München, Germany

Summary. Secure web information systems are becoming increasingly important due to rising cybercrime as well as the growing awareness of data privacy. Besides authentication and confidential connections, both data access control and navigational access control are the most relevant security features in this field. Adding such security features, however, to already implemented web applications is an error-prone task. Our approach enables web engineers to model security issues in an early phase of the development process. We demonstrate the integration for the UML-based Web Engineering (UWE) method. The approach supports the engineer by providing means to model navigational security with a plugin in a UML modeling tool. Additionally, the models can be used for the verification of web systems and security properties, such as reachability of navigation nodes in general and of those that are restricted to authorized users.

Keywords: Security, Web Engineering, Modeling, Verification.

1 Introduction

The article *Top 25 Most Dangerous Software Errors*[1] clearly shows the relevance of security aspects in software systems. The list includes items like "Improper Access Control (Authorization)", "Missing Encryption of Sensitive Data" or "Missing Authentication for Critical Function". These threats are exacerbated by the global and 7/24 accessibility of web information systems (WISs) as well as the unforeseeable range of customers. Therefore many of those systems require role dependent sessions. Login mechanisms mostly imply not only changed

* This work has been partially supported by the DFG project MAEWA II, WI 841/7-2, the EU project ASCENS, 257414, and by the EU-NoE project NESSoS, 256980.

[1] Common Weakness Enumeration CWE/SANS. Top 25 Most Dangerous Software Errors. http://cwe.mitre.org/top25/#Details, last visited 2011-02-18.

J. Grabis and M. Kirikova (Eds.): BIR 2011, LNBIP 90, pp. 239–253, 2011.

permissions regarding accessible data or activities, but also access to particular non-public areas of a website. Restricted access often goes hand in hand with the need to take care of freshness, confidentiality and integrity while transmitting data over an insecure network as the Internet.

More often than not security is added to already implemented software, i.e. too late in the development process, for example after data leaks have been detected. However, incidents like publicly accessible credit card details or personal registration data could be mostly avoided using an engineering method supporting the security concern from the beginning of the development process. In fact, web application frameworks, such as Spring, Joomla, RubyOnRails, Lift and Zope[2] offer support for the implementation of security aspects. The support varies from integrated elements in the language to specific modules or plugins that need to be explicitly installed, mainly based on access control lists (ACL). What still is missing is the possibility to handle these security topics earlier in the web application life cycle.

Our aim is to provide convenient modeling techniques that enable WIS developers not only to figure out what customers need exactly, but also how their idea of security can be specified in a concrete and intuitive manner and afterwards seamlessly implemented in any selected framework. In particular, our approach allows the developer to model security features such as access control, authentication and secure connections graphically with UML. The aim of authentication is to gain access to a protected resource. The process of authorization determines what a *subject* (e.g., a user or a program) is allowed to access, especially what it can do with specific *objects* (e.g., files) [1]. First and foremost, WISs have to protect not only data and restrict which functions can be called, but also have to take navigational access control into account, i.e. the parts of webpages that are accessible by a certain user. If only a certain user should have access, it is likely that the connection should be secured, i.e. confidentiality, integrity and freshness of the transmitted data have to be ensured. This guarantees that the data cannot be eavesdropped. In addition, replayed or altered information is recognized immediately.

Our UML extension for security aspects is a class and statechart-based approach that uses these techniques for a model-driven development of secure web information systems. Class diagrams are used to specify the content as well as the rights model, statecharts yield precise UML-based navigation models. Additionally, the statecharts can be subjected to model checking for verifying reachability of navigation nodes in general and of those that are restricted to authorized users. In fact, UWE's security features could be used in combination with any UML-based web engineering approach. The web engineer can develop secure web information systems in his favorite UML CASE tool; he only needs to include the UML-based Web Engineering (UWE) profile which extends the UML with a set of web and security features. Additional support is provided by

[2] Wikipedia: *Comparison of web application frameworks.* `http://en.wikipedia.org/wiki/Comparison_of_Web_application_frameworks`, last visited 2011-05-15.

the MagicUWE plugin for MagicDraw[3], which eases the modeling task by a set of direct accessible stereotyped elements, shortcuts and patterns.

The remainder of this paper is structured as follows: Section 2 outlines the main characteristics of web engineering approaches, in particular their implementation in UWE, and links them to security. Section 3 presents the UWE security extension focusing on navigation and data access control, where we use a simple online address book for illustration. How to work with our approach is described in Sect. 4, presenting the tool support for constructing models and their use for verification. Section 5 discusses related work. Finally, we give an outlook on future steps in the development of secure web information systems.

2 Web Engineering

Web Engineering is a specific domain in which model-based software development has been successfully applied. Existing approaches, such as OOHRIA [14], OOWS [18], UWE [12], or WebML [16] already provide well-known methods and tools for the design and development of web applications. Most of them follow the principle of "separation of concerns" using separate models for views, such as content, navigation, presentation, business processes, et cetera.

Content. The content model is used to represent the domain concepts that are relevant for the web application to be built and the relationships between them. Visualization of certain content is very often associated to a successful authentication, i.e. require a name and a predefined, correct *password* or a *digital certificate*.

User Model. A user or context model can be used to collect information needed for adaptation. A role model is a special case of user model, in which characteristics of the user groups are defined with the purpose of authorization and access control. Very often content and user model are integrated.

Navigation. The navigation model is used to represent navigable nodes of the hypertext structure and the links between nodes. The existing approaches differ in the representation of these web-specific concepts. Some see the nodes as pages of WISs; others distinguish between the idea of navigation node and page, where a page can be composed of several navigation nodes. There is also a difference in the choice of a structural or behavioral representation for the navigation structure. Navigational nodes are constrained by navigational access control rules, which means that users without permission can see only an error message or an advertisement for a less restrictive account. Furthermore, information accessed through navigation may have specific requirements on the confidentiality, integrity and freshness of data of the web application content.

Presentation. There are two trends in the specification of layout concerns in web engineering. On one hand the use of prototypes, on the other hand sketching the user interface using mock-up tools or modeling the presentational aspects. The objective is either a detailed platform-specific specification of the user interface

[3] MagicDraw. `http://www.magicdraw.com`, last visited 2011-02-20.

or the rough layout of the pages. In any case not only the static GUI widgets but also features of Rich Internet Applications (RIAs) like auto-completion in search fields, live validation of input fields, or drag & drop functionality should be represented in the presentation model. Regarding security features privacy also plays a role for the adaption of presentational aspects, e.g., private calendar entries should not be shown during working meetings.

Process. The process model aims to represent the workflows which are invoked from certain navigation nodes. The same security considerations apply as for the navigation model.

UWE strongly supports this "separation of concerns" selecting the appropriate UML diagram type and elements for each web concern and using the UML extension mechanisms, i.e. defining a set of stereotypes and tagged values in a profile. Concepts of the content and user model and their relationships are shown as classes and associations in a UML class diagram. For the navigation model UWE provides two different graphical representations: a structural visualization as UML stereotyped class diagrams and a behavioral form using UML state machines, which eases the specification of security features. UWE's presentation model is visualized like a mockup, using composite structure diagrams in which composition is visualized as nested classes and properties. Stereotypes are used for GUI widgets, and tagged values to represent RIA functionality. The process model comprises two views: first the process structure model that describes the relations between the different process classes, which are related to the navigation, and second the process flow model that shows the workflow for each process. They are represented by UML class diagrams and UML activity diagrams, respectively. In the following, we add to UWE a model for access control and extend UWE's behavioral representation of the navigation model.

3 Modeling Access Control in Web Information Systems

We integrate the modeling of access control, both for data and navigational access, into the modeling of WISs and RIAs. This integration enables the modeler to address security aspects right from the early phases of the development life cycle. We rely on standard UML modeling techniques and the definition of a UML profile extending UWE. The profile enhances class diagrams to specify a basic rights model building on role-based access control (RBAC) and state machines for modeling controlled navigation. We illustrate our approach by a secure address book application.[4] Due to lack of space, not all features of our security extension are used in this example, for further information the reader is referred to [3].

Case study. The WIS should allow registered users to create and navigate several address books and to add and retrieve contacts in them. Non-registered visitors can only read an introduction and the terms of service until they register or

[4] More information about the secure address book case study can be found at `http://uwe.pst.ifi.lmu.de/exampleSecureAddressBook.html`, last visited 2011-05-07.

authenticate themselves. Administrators cannot use the address book functionality, but they are allowed to search for users and to delete their accounts including all address books and contacts.

3.1 Basic Rights Model

The basic rights model is used to specify access control rules for domain concepts which are represented as UML classes and class instances. The Role instances from UWE's user model and content (or user model) classes, their attributes, and methods are connected with stereotyped dependencies. These dependencies, on the one hand, specify create/read/update/delete (CRUD)-rights; on the other hand, an execution dependency between a role and a method grants execution rights for the method.

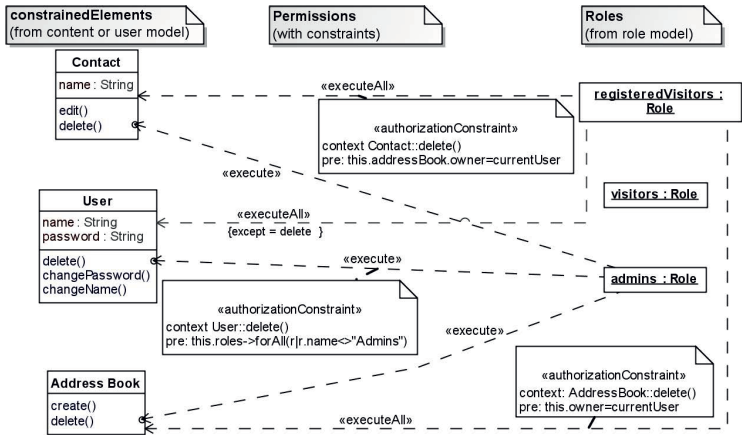

Fig. 1. Address book: Basic rights model

For the address book example, the basic rights diagram in Fig. 1 specifies execution rights on methods with dependencies stereotyped ≪execute≫ and ≪executeAll≫ (the CRUD support using ≪read≫/≪readAll≫ etc. is not shown in this example). The dependencies connect the role instances shown on the right in Fig. 1 to the methods of the content model, like Contact and AddressBook, or of the user model, such as the class User, on the left. In particular, a non-registered visitor has no execution permissions. The {except} tag for ≪. . .All≫ stereotypes allows the modeler to avoid the creation of too many dependencies. For instance, a registered visitor can execute all methods of a user object except delete. Further restrictions are added in comments stereotyped by ≪authorizationConstraint≫ in the Object Constraint Language (OCL): A registered visitor shall only be allowed to delete his own contacts and address books; an administrator shall have the permission to delete all users except other administrators. The corresponding restrictions on ≪execute≫ for Contact::delete(), AddressBook::delete(), and

User::delete() use attributes like AddressBook.owner and User.roles from the content model. The pre-defined currentUser refers to the user of the current session.

UWE's basic rights model offers a compact notation for access specifications where permissions and prohibitions can be readily read off. This is in contrast to approaches like SecureUML [13] where all permissions have to be specified separately in association classes, and exceptions cannot be expressed. However, transformations between SecureUML and our basic rights model are possible.

3.2 Navigation State Model

A navigation state model describes the navigation structure of a WIS and its behavior according to the different states. In UWE, navigation can be represented by a UML state machine: States, possibly hierarchical, represent navigational nodes, transitions the navigational links between the nodes. The UWE security profile allows to integrate navigational access control, but also session management and secure connections into the state machines specifying navigation. In particular, the navigational state model should be aligned with the access control mechanisms in the basic rights model, as e.g., a user who is not allowed to access a function of a class should be disallowed to navigate to a node that uses this functionality and vice versa.

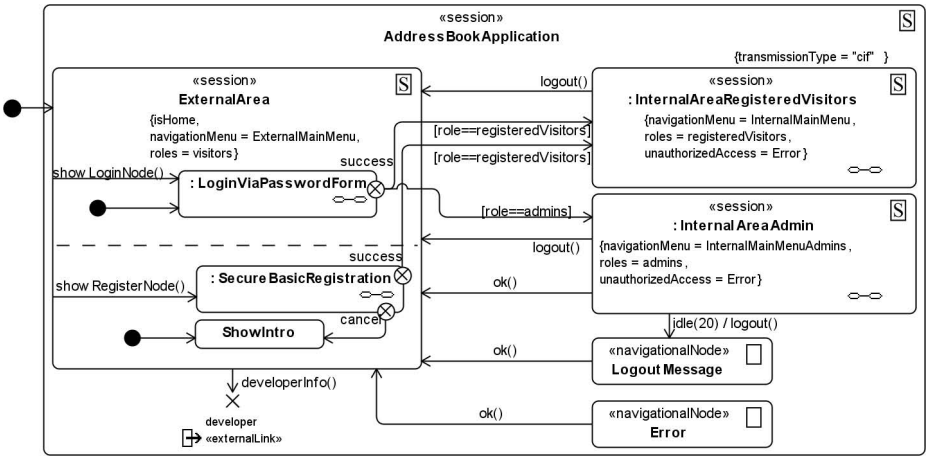

Fig. 2. Address book: Outermost navigation state model

Figure 2 depicts the main navigation state diagram for the address book example. All states are navigational nodes; when the system is in a particular state, the information and behavior offered by this state is accessible.

Navigation starts inside ExternalArea (pointed to from the outermost initial state) which is a sub-state of AddressBookApplication. Here, the sub-state machine LoginViaPasswordForm (indicated by ⋯) and state ShowIntro are entered

simultaneously (as ExternalArea shows two regions separated by a dashed line). ExternalArea is the starting point of the WIS, tagged by {isHome}. The tag {navigationMenu=ExternalMainMenu} tells that when inside ExternalArea and whichever sub-states, the user can access the actions from the navigation menu ExternalMainMenu, which include showLoginNode, showRegisterNode, and developerInfo (we omit a UML representation as a class diagram). When showLoginNode is selected, LoginViaPasswordForm is entered, when showRegistrationNode is chosen SecureBasicRegistration is entered. Both sub-state machines are, in fact, instances of the UWE security patterns offered for recurring security issues to be discussed in Sect. 4.1 (see also Fig. 4 to the left and Fig. 5).

After successful login or successful registration (leaving the two success exit points), two types of internal areas can be reached: one for the administrators and one for the registered users who want to manage their contacts. However, the subsequent area depends on the role from the role model the user takes on during login (we assume that role visitors is the default role): The guards on the transitions targeting the internal areas check the access rights. In these guards, currentUser.role is abbreviated to role.

The areas ExternalArea, InternalAreaRegisteredVisitors, and InternalAreaAdmin, as well as the super-state AddressBookApplication are distinguished navigational nodes: Their stereotype ≪session≫ (🖹) shows that context information on the navigating user is kept. The areas also restrict navigational access to them to particular roles by using the tag {roles=...}, such that, e.g., InternalAreaRegisteredVisitors can only be entered by registeredVisitors. This restriction not only protects against access through navigational transitions which should show appropriate guards, but also prohibits direct unauthorized access via a URL. Additionally, the tag {unauthorizedAccess=...} specifies which state is entered when the access rule is violated; for both internal areas this state is Error. The tag transmissionType="cif" for the session state AddressBookApplication sets the overall type of data transmission during the session to cif, providing for confidentiality, integrity, and freshness: The implementation should prevent eavesdropping, replaying, or altering of transmitted data. The transmission type is sustained also in the sub-states.

InternalAreaAdmin can be left explicitly by choosing logout from its navigation menu InternalMainMenuAdmins; it will also be left when the user stays idle for more than 20 time units after which the application will transit into the navigational node LogoutMessage. Finally, the ≪externalLink≫ (🖹) developer can be reached from ExternalArea. When this external web page is opened in a new browser window or tab, the system will still be inside ExternalArea, otherwise the WIS is left.

UWE navigational states profile. The excerpt of the UWE profile in Fig. 3 summarizes the integration of navigation and security we have illustrated for the address book example.

The basic state and state machine stereotype for navigational state models is ≪navigationalNode≫. Here, "navigational" refers to the view and the granularity of the state machines, because not all states and transitions need to

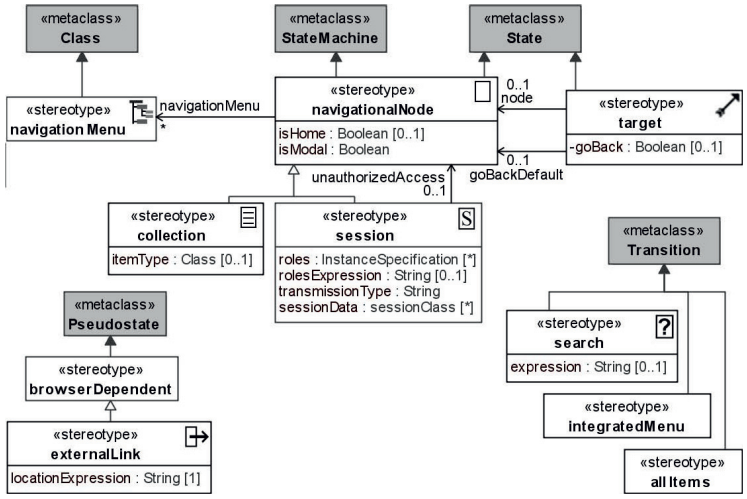

Fig. 3. Navigation states profile

represent navigational behavior. In particular, an ≪externalLink≫ is not a proper navigational state. The initial node of a WIS is marked by isHome. A ≪navigationalNode≫ can also be set as isModal, meaning that no other navigational node of the navigation model can be accessed as long as the modal node is active. Each ≪navigationalNode≫ can also refer to ≪navigationMenu≫s containing the operations which can be chosen from within the node. The ≪session≫ stereotype is derived from ≪navigationalNode≫ and thus inherits navigationMenu and isHome; a session additionally keeps sessionData and specifies role access restrictions (roles, unauthorizedAccess, and rolesExpression for more fine-grained rules) and a transmission type.

The UWE profile offers some further features for modeling navigation (cf. [3]): The tag {goBack} of the stereotype ≪target≫ (↗) allows to navigate to the state which previously had been active. Access to collections, e.g., to lists, is specially supported. Also, large menus can be integrated on transitions, which is particularly useful if each user can be associated with a set of roles.

Transformation to code. The UWE profile for integrating navigation and security in WIS is constructed in a way that transformation to code can be achieved in the near future. Such a transformation can utilize navigational access control offered by several web frameworks. The states of the navigation state model become HTML-fragments and the annotated access rules — together with UWE's role model — user role representations and rules for the web framework. For instance, the Scala-based Lift framework[5] controls navigational access via its site-map feature: A list of menu items connects all HTML fragments with the modeled access rules. Here, for InternalAreaRegisteredVistors a rule has to be stated that

[5] Lift. `http://www.liftweb.net/`, last visited 2011-05-01

the current user has to be associated with the role registeredVisitors. This is specified as an immutable variable in Scala and is used to regulate the access to every HTML fragment within this internal area.

Representation of Navigational Stereotypes in Plain UML. The precise meaning of UWE's navigational and security stereotypes and tags is defined by transformations into plain UML. Here, we only illustrate one such transformation for the running address book example; all transformation definitions can be found in [3]. In particular, the extended plain UML model resulting from applying the transformations can be subject to verification by model checking, as presented in the next section.

We translate the session tags {roles=admins} and {unauthorizedAccess=Error} for InternalAreaAdmin (see Fig. 2) into UML. When removing the tags, we have to ensure that no (sub-)state of InternalAreaAdmin can be accessed without showing role admins, either directly or by recording a URL, but these users have to be redirected to Error. InternalAreaAdmin has two sub-states for which a URL could be recorded, one for searching a user and another for deleting a user (we omit the internal state machine of InternalAreaAdmin). Figure 4 shows the result of the transformation for this particular situation.

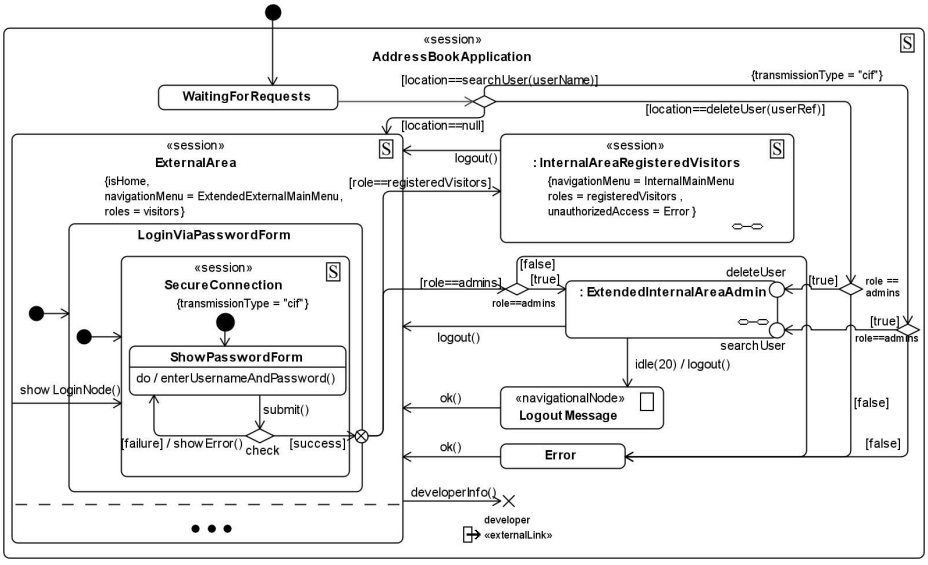

Fig. 4. Address book. Partly extended navigation state model.

If a user stores a URL, which is modeled as location variable that is set while WaitingForRequests, but is not logged in or is no longer allowed to take on the admin role, the application navigates to the Error state. (1) The two choices on the right have equal conditions, but guard the admin area from direct access.

(2) The choice in the center of the diagram protects the admin area from unauthorized access from ExternalArea (however, this is not necessary, because the guard on the transition before is strong enough). On the left of the extended state machine in Fig. 4, we have also unfolded the login pattern and we hide the registration functionality by '. . . '.

4 Working with UWE Security Models

The aim of our approach is to allow web engineers to address security aspects in an early phase of the development process. For this purpose the UWE profile includes modeling elements and diagrams specific to the web and security domain, providing a so-called domain specific modeling language (DSML). However, to further ease the web engineer's work a pattern catalogue for recurring issues and tool support for modeling and model validation are needed as well. Therefore, we added navigational patterns to the UWE profile, implemented a series of modeling supporting features as part of a plugin for the UML CASE tool MagicDraw, and offer the possibility to formally check model properties.

The applicability of the overall approach is proven by the design and implementation of a real (although simplified in the sense of focusing on security aspects) web-based hospital information system (HIS). For a detailed description and the sources the reader is referred to the web page of HospInfo[6].

4.1 Security Patterns

Patterns are a common approach to tackle the problem of repetitive tasks. "A security pattern describes a particular recurring security problem that arises in a specific security context and presents a well-proven generic scheme for a security solution." [17] We use security patterns specified as state machines that can be easily included as sub-state machines in navigation state diagrams. Typical examples are registration, authentication (login mechanisms), credential recovery (lost password), or profile configuration. We only explain the registration pattern, further examples can be found in [3].

For user registration commonly at least two things have to be checked: The user to be registered should be human; and the information the user provides has to be valid, for example the given email address should have the correct format. Another frequent requirement is to encrypt the entered data during the transmission to the server. Accordingly, the registration pattern, see Fig. 5, is modeled as a sub-state machine stereotyped as session and comprising a session state representing the secure connection by the tag {transmissionType="cif"}. Inside SecureConnection, a CAPTCHA[7] to tell computers apart from humans and the input of user data are offered. Only if both regions have been filled in

[6] HospInfo. A secure hospital information system `http://uwe.pst.ifi.lmu.de/exampleHospInfo.html`, last visited 2011-04-20.

[7] An example is Google's reCAPTCHA. `http://www.google.com/recaptcha`, last visited 2011-02-10.

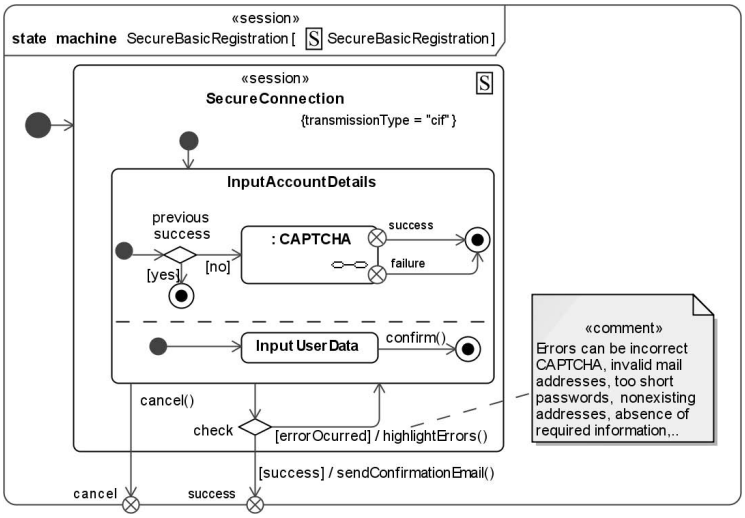

Fig. 5. Secure registration pattern

and no errors are detected by the check registration is successful. In fact, this check as well as the input data to be provided have to be customized when the pattern is applied.

4.2 Tool Support for Modeling

Tool support is crucial for the applicability of a methodology such as UWE. We decided to rely on the MagicDraw CASE tool for modeling all kinds of web applications [4]. Nevertheless, our tool concept may be adopted for other commercial and open source UML CASE tools.

MagicUWE[8] is a MagicDraw plugin for developing secure WISs with the UWE UML-profile in order to ease the modeling activities. Whenever models are created, some tasks have to be repeated over and over. Furthermore, some consistency checks and transformations are very time consuming, if executed manually. The solution is to provide plugin features that provide shortcuts for tasks like (1) inserting UWE's stereotyped elements directly from the toolbar and (2) copying UWE stereotypes and their tags between a state machine and its substate machines, (3) specifying tags facilitated by a context menu, (4) deriving the type of a substate from the stereotypes of a superstate recursively and (5) inheriting stereotypes for use cases stored in a package and (6) checking features of the models. Figure 6 depicts some of these plugin features.

An example for such a functionality is the check whether the transmission type of a secure connection is changed within nested states of the application's navigation model. This allows the modeler to see, if a service as e.g. a CAPTCHA

[8] MagicUWE. `http://uwe.pst.ifi.lmu.de/toolMagicUWE.html`, last visited 2011-05-08.

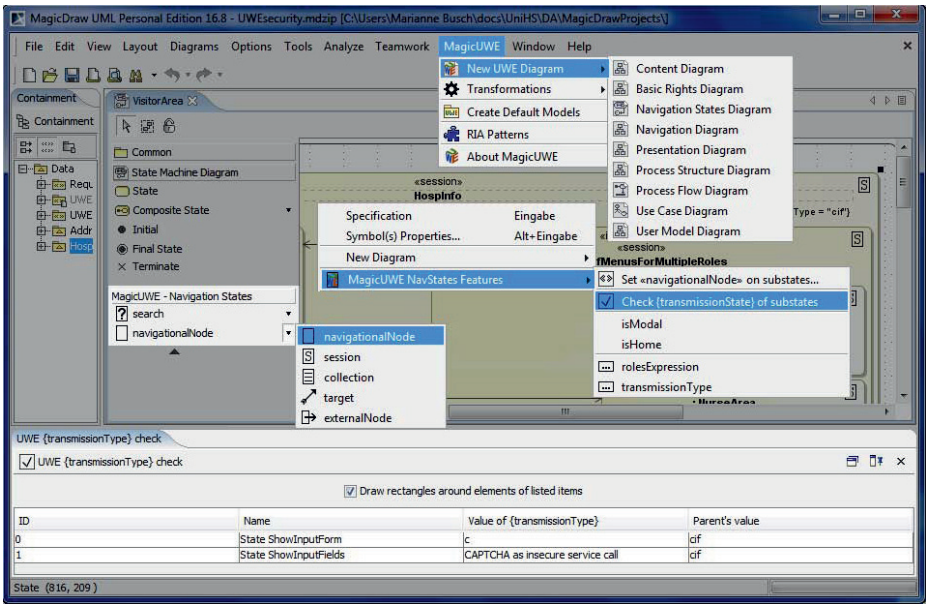

Fig. 6. MagicUWE plugin

that is used within a secure area communicates over an unencrypted connection. Without MagicUWE, all substates would have to be checked for changes by hand, which can be very time-consuming for larger models.

4.3 Validation of Models

The expressiveness and flexibility of the UWE profile and security modeling techniques makes it desirable to obtain feedback whether the model indeed satisfies the modeler's security intentions. Since UWE builds on standard UML and, in particular, UML state machines, by reducing many security modeling features to plain UML expansions, we may apply formal techniques developed for standard UML, like model checking [11,7] or theorem proving [2].

We use the UML model checking tool Hugo/RT [11] to check the UML state machine in Fig. 4, which results from the UWE navigation model, for security problems. For example, we want to ensure that no unauthorized user can enter the state ExtendedInternalAreaAdmin, i.e., whenever in this state the current user must play the role admins. In the temporal OCL extension supported by Hugo/RT this property reads:

```
G a.inState(AddressBookApplication.
              ExtendedInternalAreaAdmin) implies
    a.currentUser.role == ADMINS;
```

where `G` is the linear-temporal logic operator "always" and object `a` represents the application.

The WIS has little control on what a user may try in order to reach some location, be it that the user just follows links or that the user browses to a location directly using link guessing or previously recorded links. We thus build (currently manually) an attacking user who tries all possible interactions with the web application in all possible ways; this user is again represented as a UML state machine. Hugo/RT translates the state machines for the web application and the user, as well as the assertion into the input language of a back-end model checker, in this case SPIN [9]. SPIN then verifies that the assertion indeed holds. In fact, such a property may look quite obvious in our running example; however, the situation can get rather complicated in bigger applications, like HospInfo.

5 Related Work

This work is related to several security and web engineering approaches, mainly to those which focus on UML-based specification of secure systems.

UMLsec [10] 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 very detailed and therefore quickly become very complex. The main difficulty is the missing support of UML 2 since the provided UMLsec tools[9] and the used CASE tool ArgoUML[10] still only supports UML 1.4.

SecureUML [13] is a modeling language for the model-driven development of secure, distributed systems also based on UML. 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 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.

There is a set of approaches that address modeling of security aspects of service-oriented architectures (SOAs), such as the SECTET framework [8], the SENSORIA approach UML4SOA [6], and SecureSOA [15]. 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.

[9] UMLsec Analysis Tools. `http://ls14-www.cs.tu-dortmund.de/main2/jj/umlsectool/`, last visited 2011-03-15.

[10] ArgoUML. `http://argouml.tigris.org/`, last visited 2011-03-20.

6 Conclusions and Future Work

We have addressed access control and other security features in an early phase of the development process of web information systems. Modeling elements and patterns for RBAC, secure communication links and authentication-related processes are provided, which can be applied to navigational aspects of the web information system. UWE's security features are defined in such a way that repetitions are avoided whenever possible and security-specific modeling elements are offered according to the granularity needed for the implementation. UWE is a UML-based approach based on state machines for the representation of secure navigation and secure patterns. The UWE profile is UML-compliant, i.e., usable by any UML CASE tool. Additional comfort in the modeling process is provided by the MagicDraw plugin [4].

The applicability of our approach is proven by two case studies: a hospital information system and a secure address book. The first one covers the full WIS development life cycle, from requirements analysis, through modeling to a concrete implementation in Scala using the Lift framework. The second one was used for illustrating the advantages of a concise notation and the verification facilities of the navigation state machines introduced in this work that are checked using the Hugo/RT UML verification tool.

Future work will encompass further validation consisting in the combination of UWE's security features and other web engineering methods, such as the UML version of WebML [16]. In addition, we plan to extend the current UWE approach to cover the generation of a database scheme for the access rights. For this purpose we will consider the ongoing research work of Egea et al. [5] concerning SecureUML [13] and databases. The corresponding transformations will be implemented in our MagicUWE plugin. Another interesting issue is the separation of concerns regarding security aspects (e.g. as a list of requirements) and our navigation states model. A combination with the aspect-oriented modeling approach HiLa from Zhang et al. [19] might be promising. Furthermore, we are working on a code generator that transforms UWE navigation models to Scala, especially the page structure and the according access rights.

References

1. Anderson, R.J.: Security Engineering: A Guide to Building Dependable Distributed Systems, 2nd edn. Wiley, Chichester (2008)
2. Balser, M., Bäumler, S., Knapp, A., Reif, W., Thums, A.: Interactive Verification of UML State Machines. In: Davies, J., Schulte, W., Barnett, M. (eds.) ICFEM 2004. LNCS, vol. 3308, pp. 434–448. Springer, Heidelberg (2004)
3. Busch, M.: Integration of Security Aspects in Web Engineering. Master's thesis, Ludwig-Maximilians-Universität München (2011), `http://uwe.pst.ifi.lmu.de/publications/BuschDA.pdf`
4. Busch, M., Koch, N.: MagicUWE – A CASE Tool Plugin for Modeling Web Applications. In: Gaedke, M., Grossniklaus, M., Díaz, O. (eds.) ICWE 2009. LNCS, vol. 5648, pp. 505–508. Springer, Heidelberg (2009)

5. Clavel, M., da Silva, V., Braga, C., Egea, M.: Model-Driven Security in Practice: An Industrial Experience. In: Schieferdecker, I., Hartman, A. (eds.) ECMDA-FA 2008. LNCS, vol. 5095, pp. 326–337. Springer, Heidelberg (2008)
6. 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), 287–311 (2011)
7. Gnesi, S., Mazzanti, F.: On-The-Fly Model Checking of Communicating UML State Machines. In: Proc. 2nd ACIS Int. Conf. Software Engineering Research, Management and Applications (SERA 2004), Los Angeles (2004)
8. Hafner, M., Breu, R.: Security Engineering for Service-Oriented Architectures. Springer, Heidelberg (2008)
9. Holzmann, G.J.: The SPIN Model Checker: Primer and Reference Manual. Addison–Wesley, London (2004)
10. Jürjens, J.: Secure Systems Development with UML. Springer, Heidelberg (2004); Tools and further information, `http://www.umlsec.de/`
11. Knapp, A., Merz, S., Rauh, C.: Model Checking - Timed UML State Machines and Collaborations. In: Damm, W., Olderog, E.-R. (eds.) FTRTFT 2002. LNCS, vol. 2469, pp. 395–416. Springer, Heidelberg (2002)
12. Koch, N., Knapp, A., Zhang, G., Baumeister, H.: UML-based Web Engineering: An Approach based on Standards. In: Web Engineering: Modelling and Implementing Web Applications. Human-Computer Interaction Series, pp. 157–191. Springer, Heidelberg (2008)
13. Lodderstedt, T., Basin, D., Doser, J.: SecureUML: A UML-Based Modeling Language for Model-Driven Security. In: Jézéquel, J.-M., Hussmann, H., Cook, S. (eds.) UML 2002. LNCS, vol. 2460, pp. 426–441. Springer, Heidelberg (2002)
14. Meliá, S., Gómez, J., Pérez, S., Díaz, O.: A Model-Driven Development for GWT-Based Rich Internet Applications with OOH4RIA. In: Proc. 8th Int. Conf. Web Engineering (ICWE 2008), pp. 13–23. IEEE, Los Alamitos (2008)
15. Menzel, M., Meinel, C.: A Security Meta-model for Service-Oriented Architectures. In: Proc. 2009 IEEE Int. Conf. Services Computing (SCC 2009), pp. 251–259. IEEE, Los Alamitos (2009)
16. Moreno, N., Fraternali, P., Vallecillo, A.: WebML modelling in UML. IET Software 1(3), 67 (2007)
17. Schumacher, M.: Security Engineering with Patterns: Origins, Theoretical Models, and New Applications. LNCS, vol. 2754. Springer, Heidelberg (2003)
18. Valverde, F., Pastor, O.: Applying Interaction Patterns: Towards a Model-Driven Approach for Rich Internet Applications Development. In: Proc. 7th Int. Wsh. Web-Oriented Software Technologies, IWWOST 2008 (2008)
19. Zhang, G., Hölzl, M.: Aspect-Oriented Modeling of Web Applications with HiLA. In: Wsh. Proc. 11th Int. Conf. Web Engineering (ICWE 2011). LNCS. Springer, Heidelberg (to appear, 2011)