# Integration of Security Aspects in Web Engineering

Marianne Busch

Institut für Informatik
Ludwig-Maximilians-Universität München


Diplomarbeit

# Integration of Security Aspects in Web Engineering

Marianne Busch

| | |
|---|---|
| aufgabensteller | Prof. Dr. Wirsing |
| betreuer | Dr. Nora Koch |
| abgabedatum | 24.2.2011 |

# DECLARATION

Hiermit versichere ich, dass ich die vorliegende Diplomarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

*München, 24.2.2011*

Marianne Busch

# ABSTRACT

*Secure* web applications are becoming increasingly important due to rising cybercrime as well as the growing awareness of data privacy. Since adding security features to already existing applications can be a very time-consuming task, it is important to take security aspects into account while planning and modeling a web application.

In this thesis, the challenges of security in the web and currently existing security and web engineering approaches are investigated in order to create a comprehensive and coherent security modeling technique for web applications. Web developers should be enabled to model security aspects as authentication, access control and secure connections efficiently. For this purpose, the UML-based Web Engineering (UWE) approach, which has been developed at the Institute of Programming and Software Engineering of the Ludwig-Maximilians-Universität München is enhanced. Despite its seamless integration into UWE, the concept presented here can also be utilized independently for other UML-based modeling techniques.

In order to create a versatile approach, the functionality of the MagicDraw plugin *MagicUWE* is complemented by appropriate features. A case study of a Hospital Information System verifies the suitability for daily use of the approach as well as of the tool. To this effect, a prototype is not only modeled, but also implemented. Additionally, an address book example presents alternatives regarding the usage of modeling elements.

# ZUSAMMENFASSUNG

*Sichere* Webanwendungen gewinnen nicht nur wegen der zunehmenden Internetkriminalität an Bedeutung, sondern auch durch das steigende Bewusstsein für Datenschutz. Da Sicherheit nur mit großem Aufwand nachträglich zu einer Anwendung hinzugefügt werden kann, ist es wichtig, Sicherheitsaspekte bei der Planung und Modellierung von Webanwendungen mit einzubeziehen.

Im Vorfeld werden in dieser Arbeit zunächst die Herausforderungen von Websicherheit beschrieben und bestehende Modellierungstechniken in Betracht gezogen – sowohl im Bereich des Security-, als auch im Bereich des Webengineerings. Darauf aufbauend wird der am Lehrstuhl für Programmierung und Softwaretechnik der Ludwig-Maximilians-Universität München entwickelte Ansatz namens UML-based Web Engineering (UWE) um ein umfassendes Sicherheitskonzept erweitert, das dem Webentwickler die effiziente Modellierung von Authentifizierung, Zugriffskontrolle und sicheren Verbindungen ermöglicht. Trotz der nahtlosen Anbindung an UWE kann das Konzept auch unabhängig für andere UML-basierte Methoden eingesetzt werden.

Für die Verbreitung des neuen Ansatzes wird das MagicDraw Plugin *MagicUWE* um passende Funktionalitäten ergänzt. Das Fallbeispiel eines Krankenhausinformationssystems dient der Überprüfung der Praxistauglichkeit des Ansatzes sowie des Werkzeugs. Dabei wird besonderes Augenmerk auf die Modellierung eines Prototypen und dessen Implementierung gelegt. Ein Adressbuch als weiteres Fallbeispiel erläutert Alternativen hinsichtlich der Modellierung.

*Most interesting of all, however, is the lesson that the bulk of computer security research and the development activity is expended on activities which are of marginal relevance to real needs. A paradigm shift is underway, and a number of recent threads point towards a fusion of security with software engineering, or at the very least to an influx of software engineering ideas.*

— Ross Anderson

## ACKNOWLEDGMENTS

My thanks go to *Dr. Nora Koch* for the supervision of this thesis and her help with words and deeds. It has always been a pleasure working together with her: as a student assistant for the last four years as well as in the course of my bachelor thesis ("Projektarbeit"). I have treasured not only our lively discussions but also thousands of nightly e-mails and many phone calls.

I would like to thank *Prof. Dr. Wirsing*, who offered me the opportunity to take part in the NESSoS Kickoff located in Pisa and to attend the second meeting in Madrid. It was a rewarding experience to discuss with so many experts in person.

Furthermore, I am grateful *Dr. Matthias Hölzl* for his advice regarding literature about security and for previously suggesting several interesting subject areas in the field of software engineering.

Special thanks go to *Prof. Dr. Alexander Knapp* and *Dr. Gefei Zhang* for clarifying details and for pointing out that some errors in the UML state machine specification are of a historical nature, due to the renaming of elements.

Finally, I would like to thank my friends who proofread this thesis and my *partner* and my *family* for their understanding and in particular *Dr. Claudia Busch* for sharing her knowledge about hospital information systems.

# CONTENTS

# LIST OF FIGURES

# ACRONYMS

| | |
|---|---|
| AAC | Attribute-based Access Control |
| ACL | Access Control List |
| AES | Advanced Encryption Standard |
| API | Application Programming Interface |
| BGP | Break-Glass Policy |
| BPMN | Business Process Modeling Notation |
| CA | Certificate Authority |
| CAPTCHA | Completely Automated Public Turing test to tell Computers and Humans Apart |
| CASE | Computer-Aided Software Engineering |
| CD | Compact Disc |
| CERT | Computer Emergency Response Team |
| CGI | Common Gateway Interface |
| CRUD | create, read, update and delete |
| DAC | Discretionary Access Control |
| DFG | Deutsche Forschungsgemeinschaft |
| DICOM | Digital Imaging and Communications in Medicine |
| DNS | Domain Name System |
| DRG | diagnosis-related group |
| DRM | Digital Rights Management |
| EPR | electronic patient record |
| ERM | Entity-relationship model |
| FMC | Fundamental Modelling Concept |
| GPS | Global Positioning System |
| GUI | Graphical User Interface |
| HIS | Hospital Information System |
| HL7 | Health Level 7 |
| HMAC | Hash-based Message Authentication Code |
| HPC | Health Professional Card |
| HTML | Hypertext Markup Language |
| http | Hypertext Transfer Protocol |
| https | Hypertext Transfer Protocol Secure |
| IDE | integrated development environment |
| IEC | International Electrotechnical Commission |
| IP | Internet Protocol |
| IS | information system |
| ISO | International Organization for Standardization |
| Jif | Java + information flow |
| JVM | Java Virtual Machine |
| MAC | Mandatory Access Control |

| | |
|---|---|
| MAEWA | Model-Driven Engineering of Web Applications |
| MD | Message-Digest algorithm |
| MDSD | Model-Driven Software Development |
| NMRI | nuclear magnetic resonance imaging |
| OCL | Object Constraint Language |
| OID | Object Identifier |
| ORM | object-relational mapping |
| OSI | Open Systems Interconnection |
| OWASP | Open Web Application Security Project |
| PACS | picture archiving and communication system |
| PC | Personal Computer |
| pdf | Portable Document Format |
| PGP | Pretty Good Privacy |
| PKI | Public Key Infrastructure |
| POM | Project Object Model |
| RBAC | Role-Based Access Control |
| REST | Representational State Transfer |
| RIA | Rich Internet Application |
| RIS | radiology information system |
| S/MIME | Secure / Multipurpose Internet Mail Extensions |
| SHA | Secure Hash Algorithm |
| Sif | Servlet Information Flow |
| SMS | Short Message Service |
| SOA | Service-Oriented Architecture |
| SSL | Secure Sockets Layer |
| SSO | Single Sign-on |
| TAN | Transaction Authentication Number |
| TLS | Transport Layer Security |
| TV | television |
| UML | Unified Modeling Language |
| URL | Uniform Resource Locator |
| UTF | Unicode Transformation Format |
| UWE | UML-based Web Engineering |
| WebML | Web Modeling Language |
| XMI | XML Metadata Interchange |
| XSS | Cross-site scripting |

# 1 INTRODUCTION

The article *Top 25 Most Dangerous Software Errors*[1] shows the relevance of security aspects in software systems. The list includes flaws like "Improper Access Control (Authorization)", "Missing Encryption of Sensitive Data" or "Missing Authentication for Critical Function". Those who are aware of all this will not find it surprising to read about data leaks in web applications almost every day in the news. Incidents like publicly accessible credit card details or personal registration data could be avoided for the most part by using a web engineering method that supports security aspects, e.g. secure connections and user management. This means security can be modeled in diagrams additionally to other features of up-to-date web applications. It is our intension to provide convenient modeling techniques that enable developers to figure out the actual needs of the customers and that allow them to implement their idea of security.

*Are current web applications secure enough?*

Web and security engineering have been two rather separated research areas so far. At the Institute of Programming and Software Engineering of the Ludwig-Maximilians-Universität München, web engineering techniques have been investigated for more than a decade. The UML-based Web Engineering (UWE) modeling method resulted from this research. Two consecutive Deutsche Forschungsgemeinschaft (DFG) projects[2] have leveraged both the research and the tool support for UWE.

It has become increasingly apparent that security engineering cannot be separated from web engineering any longer, because security plays an important role in almost all advanced web applications. Therefore, this work aims at integrating the various security aspects in web engineering. Our approach, called *UWEsecurity*, can be combined with any web engineering method, although this diploma thesis focuses on the combination with UWE.

*bringing together security and web engineering*

The aim of *UWEsecurity* is to model authentication, access control and secure connections graphically. First of all, it is important to provide a navigation model that supports role dependent sessions, because login mechanisms mostly do not only imply changed 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 fresh-

1 Common Weakness Enumeration CWE/SANS. Top 25 Most Dangerous Software Errors. http://cwe.mitre.org/top25/#Details, last visited 2010-11-26

2 MAEWA I and MAEWA II (Model-Driven Engineering of Web Applications). http://uwe.pst.ifi.lmu.de/infoMAEWA.html, last visited 2010-10-26

ness, confidentiality and integrity while transmitting data over an insecure network as the Internet. Instead of reinventing the wheel while modeling internal details for every application, the use of approved web concepts like Transport Layer Security (TLS) connections have to be considered.

We do not only incorporate Unified Modeling Language (UML) based modeling techniques, but also security patterns (for example for different authentication variants) in order to ease the development of secure web applications for the web engineer. According to that, our *UWEsecurity* models are designed to be clear and precise so the web developer does not loose himself in confusing diagrams, but still has the possibility to specify details as well. This implies that the users can decide on their own, if e.g. the definition of access control is important for all elements or only for particular ones.

*NESSoS project*

Beyond this work, the Institute of Programming and Software Engineering has been involved in the NESSoS[3] project since October 2010. NESSoS is an EU project that is concerned with engineering secure future Internet software. We will contribute particularly to the integration of various methodologies, similarly following the idea of linking different security aspects and tools.

*description of contents*

This thesis consists of four parts: The first part provides some background information regarding security aspects of web applications (chapter 2) and examines related work in the field of security modeling and web engineering methods (chapter 3).

The second part introduces our security engineering method *UWEsecurity*. In chapter 4 (security engineering for web applications), the UML profile is defined. Based on this profile, chapter 5 describes the realization of modeled security features in practice.

In the third part, our case study proves the applicability of our approach. This is achieved by the design (chapter 6) and implementation (chapter 7) of a prototype Hospital Information System (HIS). In order to facilitate the modeling process, new functionalities are introduced in our MagicDraw Plugin *MagicUWE*, as described in chapter 8.

Finally, part iv summarizes the results (chapter 9) and provides an outlook on future work (chapter 10). Subsequently, appendix A presents the modeling of an address book case study and appendix B lists the files that are stored on the attached CD.

3 NESSoS is the abbreviation of Network of Excellence on Engineering Secure Future Internet Software Services and Systems. http://www.nessos-project.eu/, last visited 2011-02-16

# Part I

# SECURITY, WEB APPLICATIONS AND SOFTWARE ENGINEERING APPROACHES

# 2 SECURITY ASPECTS OF WEB APPLICATIONS

The main question of this work is, how to make the modeling of security aspects easier for an web engineer and facilitate the definition of model transformations in a model-driven approach. Before caring about the 'how', we have to define the required security features and discuss their importance in the context of web applications. The technical implementations are also mentioned and a couple of sections finish with additional roles and responsibilities one might bear in mind while planning the integration of a security aspect. This chapter provides a brief overview of security concepts such as authenticity, confidentially, integrity, access control, non-repudiation, freshness, privacy and secure information flow, and discusses their relevance in web applications.

## 2.1 Entity Authenticity and Authentication

*Authentication is the act of confirming something as authentic*

In many applications the first thing users have to do is to prove who they are. This process is called authentication and the aim is to archive a state, in which the system can be sure that the person in front of the computer is most likely authentic, i.e. not taken for someone else.

An example would be the login form of online bulletin boards: it allows to establish an own profile and enables the mapping of posts to users. In other scenarios, the user has to show not only who he is, but also that his e-mail address actually belongs to him (e.g. by clicking at a confirmation link, which had been sent to that address before). Another possibility is to register a cell phone number, which is validated by sending an SMS with a Transaction Authentication Number (TAN), which has to be typed in a validation field.[1]

Using web applications the user usually has to provide a name and a predefined, correct password (knowledge) or a digital certificate[2] (property) for a successful authentication.

The advantage of a *Public Key Infrastructure (PKI)* is that certificates can contain a valid e-mail address and additional information, like the real name in the identification card. The disadvantage is that there are not many Certificate Authorities (CAs) that

1 An example can be found during the registration for E-Postbrief. https://adresse-sichern.epost.de/, last visited 2010-09-27

2 An example is the "Certificate Login" of CAcert. https://www.cacert.org/, last visited 2010-09-10

issue free public key certificates. Consequently, certificates are not widely-used for login purposes.

*Passwords* are easier to create than certificates, but the problems with them can be summarized as "choose a password you can't remember, and don't write it down"[2, page 33] (because it should be hard to find it out, either by guessing or by automated testing). To reduce the risk of password theft even more, it should often be changed and the same one must not be used for different services.

*Password recovery* sometimes requires answering questions, e.g. "your mother's maiden name", but this is considered as a design error in [2, page 37], especially because it is easy to find out and the user can't change it as frequently as passwords should be changed. Therefore, storing an e-mail address or a mobile phone number is more popular. If someone has forgotten his password but not the user name, he or she can receive an e-mail or a SMS message with a new, generated password.

There is a trend to use Single Sign-on (SSO) with OpenID.[3] In this way, the user can authenticate himself once with e.g. a Google account[4] and later on connect to other web services (e.g. Zoho[5]). The authorized access can be withdrawn[6], but it is easy for a fisherman to build a website which imitates an OpenID login form. In IT security, a fisherman is someone who steals data by pretending to be someone else.

*Two-factor authentication*[7] is more secure, as the users have to provide two types of credentials, like entering a password and a kind of TAN. Some systems also use biometrical data, e.g. the sound of a human voice or a fingerprint or the user has to provide a smart card. This is not common regarding web applications, because the users often need to be mobile and it is impractical to take scanners or card readers with them. However, a new approach is google's two-step authentication[8] that uses an app for the smart phone called Google Authenticator. It generates short codes that have to be entered additionally to the password. For usability reasons it can be configured that the additional code has to be entered only once a month for permanently used PCs. An overview of identification and authentication patterns can be found in [50, page 63].

3 Wikipedia: OpenID. http://de.wikipedia.org/wiki/OpenID, last visited 2010-09-16

4 Google Accounts. Login. https://www.google.com/accounts/Login, last visited 2010-09-16

5 Zoho. http://www.zoho.com/, last visited 2010-08-20

6 Google Accounts. My Account. https://www.google.com/accounts/IssuedAuthSubTokens, last visited 2010-09-16

7 Wikipedia: Two-factor authentication. http://en.wikipedia.org/wiki/Two-factor_authentication, last visited 2010-09-05

8 Google 2-step verification. https://www.google.com/accounts/SmsAuthConfig, last visited 2011-02-18

## 2.2 Data Origin Authenticity

*Sometimes it is important to know who created a message*

Data origin authenticity "means to secure the information on the message origin"[23, page 43]. The origin can be a system part as well as a person.

An example would be an online bank login form. The users really want to be sure that the web page has been delivered by their bank and not by a fisherman, who wants to collect their credentials.

The technical solution for the web is using X.509 server certificates and a secure connection over Transport Layer Security (TLS) or Secure Sockets Layer (SSL) and the Hypertext Transfer Protocol Secure (https). *X.509* is a standard for a PKI and the certificates guarantee that the website belongs to a certified domain. In this case, a DNS name is part of the certificate and common browsers deploy a list of root certificates of CAs in order to simplify the verification. *TLS* (the successor of SSL) is a cryptographic protocol for secure transit at the transport layer of the OSI model. It comprises e.g. data confidentiality, integrity and authentication with X.509 certificates.

| Protocol | Info |
|---|---|
| TCP | 52701 > https [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=2 |
| TCP | https > 52701 [SYN, ACK] Seq=0 Ack=1 Win=4356 Len=0 MSS=1452 WS=0 |
| TCP | 52701 > https [ACK] Seq=1 Ack=1 Win=17424 Len=0 |
| SSL | Client Hello |
| TLSv1 | Server Hello, Change Cipher Spec, Encrypted Handshake Message |
| TLSv1 | Change Cipher Spec, Encrypted Handshake Message, Application Data |
| TCP | https > 52701 [ACK] Seq=139 Ack=876 Win=4559 Len=0 |
| TLSv1 | Application Data |
| TLSv1 | Application Data |
| TCP | 52701 > https [ACK] Seq=876 Ack=597 Win=16828 Len=0 |
| TCP | [TCP segment of a reassembled PDU] |
| TCP | [TCP segment of a reassembled PDU] |
| TCP | 52701 > https [ACK] Seq=876 Ack=3501 Win=17424 Len=0 |
| TCP | [TCP segment of a reassembled PDU] |
| TLSv1 | Application Data, |
| TCP | 52701 > https [ACK] Seq=876 Ack=5380 Win=17424 Len=0 |
| TLSv1 | Application Data |

Figure 2.1: Wireshark. TLS communication

The communication can be tracked with a network protocol analyzer like Wireshark.[9] Figure 2.1 shows that the German online bank `Comdirect` uses TLS for their https login page.[10]

## 2.3 Data Confidentiality

The International Organization for Standardization (ISO) defines confidentiality as "ensuring that information is accessible only to those authorized to have access".[11] Its goal is that a thief is not able to decode any intercepted data.

9 Wireshark. http://www.wireshark.org/, last visited 2010-09-21

10 Comdirekt. Persönlicher Bereich - Login. https://kunde.comdirect.de/lp/wt/login, last visited 2010-09-21

11 ISO 17799

*Confidentiality is one of the cornerstones of information security (often summarized as confidentiality, integrity and availability)*

The areas of application are manifold: submitted Credit Card information should not be visible to everyone. Likewise, bank transfers are intended to remain a banker's secrecy.

Cryptography deals with the en- and decryption of messages in order to hide information during the transmission over the 'insecure' Internet. Web applications often use an https connection that wraps up different kinds of encryption, depending on the requirements of computation speed and on the level of security. The security settings are negotiated in a so called *cipher suite* at the beginning of a TLS connection. `Comdirect` uses https with cipher `AES256-SHA AES(256)`, which can be found out using a service called serversniff.[12] AES stands for Advanced Encryption Standard (a symmetric-key encryption) and SHA is a Secure Hash Algorithm, ensuring *integrity*, as described in the following section.

It is crucial to transmit authentication data, e.g. passwords, in a confidential way to avoid identity theft. It is not always necessary to implement an own login form, as common browsers support http *digest* access authentication and display an appropriate input box.[13] It has to be kept in mind that *basic* access authentication is only encoded, not encrypted. *Encoding* algorithms are functions, which transform an input into another one. The result may not be human readable, but the input can be restored by an inverse function. *Encrypted* information instead requires a key (e.g. a password) for data retrieval and of course the knowledge which algorithm has been previously applied.

## 2.4 Data Integrity

*Data integrity makes sure that a message remains unchanged*

No matter if someone deals with encrypted information or not, he sometimes want to be sure that nobody has altered the data. Data integrity means to enable "recipients to verify that the data has not been modified".[14]

This is not only important for downloading unchanged files, but also for sending messages. For example, if a user places a bid on eBay[15] and wants to pay a maximum of 10 Euros, the system has to care about the integrity of this value, otherwise one of the routers may be compromised and might forward the bid containing a multiple of the intended price. It would be hard

12 Serversniff. http://serversniff.de/sslcheck.php, last visited 2010-09-21

13 Wikipedia: Digest Access Authentication. http://en.wikipedia.org/wiki/Digest_access_authentication#Browser_implementation, last visited 2010-09-21

14 Preventing theft and unauthorized modification of electronic data with new ISO/IEC standard. http://www.iso.org/iso/pressrelease.htm?refid=Ref1221, last visited 2010-09-19

15 eBay. http://www.ebay.com/, last visited 2010-08-20

to demonstrate that he is not the one, who entered the exorbitant value.

*hash functions*

A cryptographic hash function is used to ensure data integrity, therefore the algorithms are designed to make it (almost) impossible to:

- find a message that has a given hash
- find different messages with the same hash
- modify a message without changing its hash (this feature is called *avalanche effect*),

No ideal hash functions exist, but there are some convenient ones which are good enough for daily use. For example SHA1 or MD5 checksums are used especially for downloads. They at least have to be stored separately, because of man-in-the-middle attacks where a third party is lying in-between the communication path of the client and the original server. In this case the client may receive bogus files as well as recalculated checksums. Hence it is more secure to use TLS at least for the transmission of the hash, as the processing time for the encryption of every download can be tremendous. It remains to add that there is no built-in check for downloads, so that users are responsible for the check by themselves.

To get a secure hash for the transmission of website content, e.g. Hash-based Message Authentication Code (HMAC) can be used. The additional usage of a secret key prevents man-in-the-middle attacks. HMAC is applied within the TLS protocol.[16] Even if MD5 and SHA1 are considered to be sufficiently secure, some sort of attacks can be successful anyway, e.g. a *collision attack* can occur. That means two identical MD5 hash values are constructed deliberately as for instance to request two interchangeable certificates. One of them might be approved by a CA and the other one can be used secretly by an attacker. [47, 48]

## 2.5 Access Control and Authorization

*Access control enables authorities to control access to resources of information systems*

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 do, especially what he can do with specific *objects* (e.g. files). Several access control techniques[17] exist:

ATTRIBUTE-BASED ACCESS CONTROL (AAC) Attributes as the user's geolocation are used (which may be estimated by

16 Wikipedia: HMAC. http://en.wikipedia.org/wiki/Hmac, last visited 2010-09-20

17 Wikipedia: Access Control. http://en.wikipedia.org/wiki/Access_control#Access_control_techniques, last visited 2010-09-20

looking up the IP address, because Global Positioning System (GPS) is no standard equipment and not implemented in all browsers).

DISCRETIONARY ACCESS CONTROL (DAC) Every object has an owner who can determine the access policy for all his objects, e.g. the objects he created by himself.

MANDATORY ACCESS CONTROL (MAC) The system determines all access policies and they can be very fine-grained. It has been one of the first access control techniques which have been used for multi-level secure systems (as explained in more detail in section 2.9).

ROLE-BASED ACCESS CONTROL (RBAC) This access policy also is determined by the system, but the structure is predefined: several roles can be added to users and each role is associated with a set of permissions. Additional constraints may be applied on top of RBAC, e.g. to grant access only for a certain period of time or to realize the Break-Glass Policy (allowing logged access in an emergency – a requirement for several medical applications). [18, page 37] RBAC is a widely used approach, for that reason our approach is built upon it.

Moreover, *digital tickets* "guarantee certain rights of the ticket owner". [12] They are widespread in the web and mainly applied for gift cards (e.g. amazon[18]) and voucher codes.[19]

Usually, web applications use a kind of role-based security system. Examples are the different user groups in web applications which are usually connected either with the amount of money the users pay[20], or with the status they have gained, e.g. by writing articles or by gaining points within a community.[21]

Technically, access control is realized by maintaining Access Control Lists (ACLs) or a set of rules and is enforced by a *reference monitor*, which is an element (e.g. a part of an operating system) that controls access to subjects. A reference monitor has to be tamper-proof and no subject can bypass it in order to gain unobserved access to an object. Regarding web applications, there is often a difference between limited access and full access with

18 Amazon. Gift Cards. http://www.amazon.com/gp/gc/ref=topnav_giftcert, last visited 2010-09-27

19 An example is http://www.vouchers.org.uk/, last visited 2010-09-27

20 An example is Last.fm's normal user vs. subscriber. http://www.last.fm/, last visited 2010-07-08

21 An example is the assurance system of CAcert. Users can meet each other to assure the correctness of their names in their identity card. More experienced assurers can confer more points and when a user has gained a certain amount of points, he get a certificate containing his name, become an assurer himself etc. https://www.cacert.org/index.php?id=19, last visited 2010-09-10

errors. The former only allows access to pages where the content is authorized for the authenticated user. The latter means that one is allowed to navigate to a page, but the content will not be presented, instead an error message, a login form or a remark is shown (or an advertisement[22] is displayed, if the content is available for a charge). [50, chapter 9.5 and 9.6]

## 2.6 Non-repudiation

*Non-repudiation provides undeniable evidence for a particular action*

Non-repudiation means that a user or a system cannot deny an action afterwards. A distinction is drawn between "non-repudiation of origin" and "non-repudiation of receipt", according to the moment the action is captured. [39, page 7]

It is difficult to realize non-repudiation for web applications, but strictly speaking every online bank and every online shop should provide it. In practice, "non-repudiation of receipt" is 'ensured' by providing e.g. a confirmation e-mail. The problem is that most of those e-mails are not even signed (e.g. by using Secure / Multipurpose Internet Mail Extensions (S/MIME) or Pretty Good Privacy (PGP)) and therefore can also be imitated. We have not found an example implementation for "non-repudiation of origin". Presumably this is not considered, as the users are self-responsible for ensuring that they receive a confirmation.

The most common way to provide non-repudiation is through the use of digital signatures in order to authenticate the sender and to keep the data unaltered. Implementations e.g. for web services include a Trusted Third Party which is responsible for logging and forwarding of messages as well as for logging "proof of receptions" (cf. [18, page 128 ff.]).

We may keep at the back of our mind that SSL/TLS does not provide non-repudiation. [51, page 13] Surprisingly, it seems not to be necessary to implement a real waterproof system for non-repudiation regarding jurisdiction. In our opinion there is room for improvement.

## 2.7 Freshness

*"Freshness (cryptography) - certainty that replayed messages in a replay attack on a protocol will be detected as such" (Wikipedia)*

The actuality of an action like a data transmission is important, as otherwise it could also be a replay of a communication which has been eavesdropped before.

The effect is quite evident: a user usually does not want to pay an item which was ordered last week more than one time.

So called replay attacks can be avoided using session tokens, nonces or timestamps. In TLS a combination of timestamps and

22 An example is StayFriends http://www.stayfriends.de/, last visited 2010-09-10

nonces is used. [14, page 38] Session tokens and nonces are often randomly chosen from a set of numbers and it has to be ensured that the possibility is low for guessing a valid reply. This can be achieved e.g. by increasing the size of the set or by using different keys for the encryption of the answer to prove not only the actuality but also the identity of the communication partner.

## 2.8 Internet Privacy

*Privacy means to reveal personal information about oneself selectively*

Internet Privacy "involves the exercise of control over the type or amount of information that persons reveal about themselves on the Internet and who may access such information".[23]

The claim for a *right to privacy* has caused considerably long pages where users can adjust their privacy settings (e.g. Facebook[24]). In spite of that, providers of social networks can never guarantee that no data theft will occur in the future, either by mistake or by an attack.

Another topic is the tracking of users with http cookies and other techniques as e.g. Flash Cookies and HTML5 storage.[25] Users often do not know which data is collected and have no overview which way their data is flowing.

Technically, the simplest way is to implement access control and to allow users to determine the usage of their data. Some enforced settings are convenient, like never publishing bank details. Especially in social networks, a kind of multilayer security is commonly used, in order to differ between public- and private data, and data only a special group of people is allowed to access.

## 2.9 Secure Information Flow

*Secure Information Flow protects data of several security levels*

Even if confidential data is stored and protected with an efficient access control system, how can users be sure that their data is always processed in a secure way, which does not lead to information leaks? Roughly speaking, secure information flow tries to control where the data goes and checks that no authenticated user forwards confidential data to a less confidential area. The other way around is interesting as well: in some systems, no insecure/low value should influence a calculation of secure/high values. [23, page 23]

Multilevel security is a precondition for secure information flow. It is defined as "the application of a computer system to process information with different sensitivities (i.e. at different

23 Wikipedia: Privacy. http://en.wikipedia.org/wiki/Internet_privacy, last visited 2010-09-20

24 Facebook. http://www.facebook.com/, last visited 2010-10-01

25 Evercookie. http://samy.pl/evercookie/, last visited 2010-10-15

security levels)".[26] The amount of categories can vary from only two (low and high) to a couple of categories (e.g. self defined ones as in Security-Enhanced Linux[27]).

An example of a kind of secure information flow in a web application is the digital library of Munich.[28] All the media can be borrowed for seven days and Digital Rights Management (DRM) (e.g. of pdfs or films) prevents the data from being distributed illegally. It is evident that access control and secure information flow always have to go hand in hand. In our case there are only two levels of security: borrowed and non-attached. To the regret of many authors, there are always ways of copying data which can also be read by a human eye (e.g. by taking screenshots).

There are only few examples where secure information flow is actually implemented. Researchers invented Java + information flow (Jif), "a security-typed programming language that extends Java with support for information flow control and access control"[29] and a software framework for web applications, called Servlet Information Flow (Sif) [8], but it seems not to be widely-used so far. The reason may be the diversity of the problem of secure information control; the issues to be covered are:

*the diversity of secure information control*

- who (user or other system) should have access to the data (*entity authenticity*), including ways of authentication and enforcement of *access control*
- how the data is transferred (*data origin authentication, confidentiality, integrity, freshness* and maybe *non-repudiation*)
- how the data should be stored (presumably encrypted)
- how to deal with the data within each program using it and to which extent the users can handle data in insecure environments (e.g. their browser, printing it or copying it)

If all this would be possible in a convenient way for all web applications, *privacy* would be no problem, as with a suitable logging system, everyone will be able to track the own data packets. Due to usability aspects, this would be very cumbersome, because every snippet of data would have to carry a DRM tag that includes a note of the origin and the security level of the data. Nevertheless, such a system could make sense for critical applications as HISs. In this case Health Professional Cards (HPCs) could be used to authenticate the physician and to authorize

26 Wikipedia: Multilevel security. http://en.wikipedia.org/wiki/Multilevel_security, last visited 2010-09-24

27 Fedora SELinux. Security context. http://fedoraproject.org/wiki/SELinux/SecurityContext, last visited 2010-10-05

28 Virtuelle Münchner Stadtbibliothek. http://www1.onleihe.de/muenchen, last visited 2010-09-01

29 Jif. http://www.cs.cornell.edu/jif/, last visited 2010-09-19

him to read some admission notes within the application of the Hospital Information System (HIS) without the possibility to copy parts of the text. This inhibits data leakage via copy/paste into insecure applications as e.g. email.

## 2.10 Roles and their Responsibilities

The topic of data security requirements cannot be restricted to the areas mentioned above. A secure Rich Internet Application (RIA) also requires foresighted web developers as well as full-fledged system administrators. Last but not least, the user should not be moved out of the focus. IT-Service-Management goes beyond of the scope of this work, but we would like to mention that Information Security Management is a part of ISO/IEC 20000 (further reading may emanate from the specification itself[30]).

### 2.10.1 *System Administrator*

Besides keeping track of the logfiles and installing updates and fixes, system administrators who care about web security preferably have to know what goes on in the security domain, e.g. by subscribing to Computer Emergency Response Team (CERT) advisories[31] and general security news.[32]

To avoid misconfiguration, they should be trained in configuring security aspects of software systems they use, e.g. in *Apache Security* [44].

The pattern community invented many patterns in order to help considering all important issues while planning and maintaining a secure system. Some patterns seem to be simple, but can be very effective, like the "document the server configuration" pattern, which name stands for its intention. [25] A more common one is the "distrustful decomposition", claiming to "move separate functions into mutually untrusting programs". [10, chapter 2.1] For example, a system administrator has to make sure that the web server runs CGI scripts with the permissions of several restricted users instead of the root's permissions.[33]

30 ISO/IEC 20000. http://www.iso.org/iso/catalogue_detail?csnumber=41332, last visited 2010-09-26

31 DFN-CERT. https://portal.cert.dfn.de/adv/archive/, last visited 2010-09-20

32 Heise Security. http://www.heise.de/security/, last visited 2010-09-20

33 An example would be the suEXEC support of the Apache HTTP server. http://httpd.apache.org/docs/2.2/suexec.html, last visited 2010-09-28

### 2.10.2 *Web Developer*

Secure programming for the web not only comprises gaining knowledge about the deployed web framework and programming language, but also about common security risks. Some of the latter are universal, e.g. account hijacking and code injection, especially for cross-site scripting. [10, chapter 4.5]

*Code injection is the exploitation of a bug that can be exploited by processing malicious input of users.*

Other risks are specific for each language and developers should be familiar with their language to avoid momentous bugs (cf. the save data buffer pattern in C [17]) and to be able to deal with common security libraries. Appropriate information is available for almost every kind of technology, examples are Ruby on Rails[34], Lift[35] and Java[36] (German readers can also refer to [52, chapter 26]).

For both, the system administrator and the web developer, the Open Web Application Security Project (OWASP) can be of help. OWASP tries "to make application security visible, so that people and organizations can make reasonable decisions about true application security risks".[37] Furthermore, they are providing several tools e.g. to check an application for known vulnerabilities (cf. skipfish[38]). If there is time, it also may be interesting to practice with the google code lab project Gruyere.[39] The online portal "Build Security In"[40] is the most comprehensive compilation of articles and references about software security, we have found. Even if it is not entirely up-to-date, it is a good starting basis and a glance at the associated CAPEC[41], classifying attacks, is worth one's while. The classification of the mechanisms of attacks provides a realistic overview what might be risky and therefore should gain attention.

### 2.10.3 *User*

Unfortunately, security engineering is such a broad subject that it is not enough to cut it down to the technical aspect. This means

34 Ruby on Rails Security Project - The Book. http://www.rorsecurity.info/the-book, last visited 2010-09-25

35 Lifts Security. http://wiki.github.com/dpp/liftweb/lifts-security, last visited 2010-09-25

36 CERT. Secure Coding Standard for Java. https://www.securecoding.cert.org/confluence/display/java/, last visited 2010-09-27

37 OWASP. http://www.owasp.org, last visited 2010-09-25

38 Skipfish. Web application security scanner. http://code.google.com/p/skipfish/, last visited 2010-09-25

39 Web Application Exploits and Defenses. http://google-gruyere.appspot.com/, last visited 2010-09-25

40 Build Security In. https://buildsecurityin.us-cert.gov, last visited 2010-09-26

41 Common Attack Pattern Enumeration and Classification. http://capec.mitre.org/, last visited 2010-09-26

the developer has to consider not only the stability of all the underlying pieces of software but also psychological aspects of the users.

*Cybercrime would be less lucrative, if the users were better informed*

"Absolutely secure" may be impossible, but it is always a game of strategy to rate risks and to take countermeasures. It is worth to mention Ross Anderson's book "Security Engineering" [2]. Chapter two – "Usability and Psychology" – which is an excellent return to the border of technical security: the human behavior.

Examples are the use of a browser's password database without turning encryption on, social engineering (e.g. asking someone to reveal a password) or shoulder surfing (watching over someone's shoulder in order to observe the typing). Often, the design of the application fosters that kind of errors, e.g. when the users have to type their password as often as in Windows Vista and Windows 7, they might get used to it and type it in every input box that pops-up, without taking a closer look.

# 3 RELATED WORK

This work is based on several security and web engineering methods. We decided to rely on the Unified Modeling Language (UML)[37], which has gained acceptance as the de facto standard for modeling software systems. Nevertheless, the following sections also provide an overview of non-UML-based work to make comparisons possible.

## 3.1 Security Modeling

Several methods of security modeling exist. Some are very formal and almost all are designed for a small scope of application only. This section gives an overview of graphical (mainly UML) and pattern-based modeling methods and each subsection finishes with a short comparison with our approach.

### 3.1.1 *UMLsec*

UMLsec [23] is an extension of UML with emphasis on secure protocols. Initially it was based on UML1.4 and even if it is gradually adapted to UML2, the provided UMLsec tools[1] and the used Computer-Aided Software Engineering (CASE) tool ArgoUML[2] still only supports UML1.4. Nevertheless, the UMLsec tool suite was transformed from a web based system to an ArgoUML plugin in a recent master thesis. [24] The disadvantage is that it is not planned to integrate UML2 in ArgoUML. Theoretically, XMI files can be used for exchange of UML projects between several UML CASE tools, but in practice XMI exports e.g. from MagicDraw[3] can often not be imported correctly to other tools like ArgoUML.

UMLsec is used for modeling:

FAIR EXCHANGE in activity diagrams. A {start} tag defines actions and if one of them had been executed, one of the actions mentioned in the {stop} tag have to be executed later on. The intention is to provide e.g. a fair exchange between a customer and a business.

RBAC is modeled by denoting activity diagrams with the stereotype «rbac» (a stereotype is a UML notation). Tags that

1 UMLsec Analysis Tools. http://ls14-www.cs.tu-dortmund.de/main2/jj/umlsectool/, last visited 2010-08-10

2 ArgoUML. http://argouml.tigris.org/, last visited 2010-08-02

3 MagicDraw. http://www.magicdraw.com/, last visited 2010-08-02

belong to this stereotype: {role} lists available roles, {right} is a list of pairs, the first entry stands for an actor and the second value the assigned role; the value of {protected} is a character string, which is equal to the constrained activity name.

GUARDED ACCESS allows to model the usage of a system like the Java security architecture, where several objects are guarded. For that UMLsec primarily connects the guarded objects with the guards, using the stereotype «guarded» with the tag {guard=Guard-Class}.

AUTHENTICITY FRESHNESS, SECRECY & INTEGRITY are data security properties of attributes that are specified as tags of the stereotype «critical», as mentioned in the following example. Jürjens often refers to 'confidentiality' as 'secrecy'.

SECURE INFORMATION FLOW is based on multilevel security as explained in section 2.9. Additionally, not only the two-tier level of information has to be set (using {high} that also is a tag of the stereotype «critical»), but also a state machine has to be designed in order to show possible information flows.

An example of a UML object with UMLsec annotations is depicted in figure 3.1. It is a small part of a case study about common electronic purse specifications as described in detail in [23, chapter 5.3]. The stereotype «critical» is used together with the tag {secrecy} to indicate that the attribute $K_p^{-1}$ should be handled in a confidential way. The tags {fresh} and {integrity} are also applied to attributes, which are merely represented in a style that is more influenced from the protocol specification symbols than from the standard UML rendering.

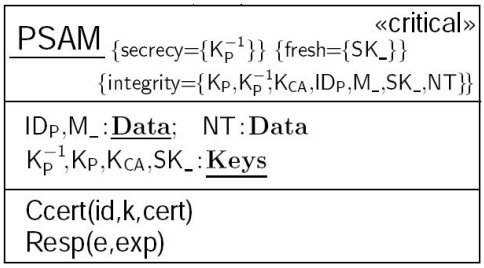


Figure 3.1: UMLsec example [34, excerpt of figure 5]

It is noticeable that UMLsec models quickly become very complex and this complexity may be acceptable and rather comprehensible in comparison to hundreds of pages of a formal specification of a security protocol. But it is comparatively impractical to apply UMLsec for modeling that does not have the main focus on details of security implementations. That is the reason why we see UMLsec as a supplement and hence do not use it for our approach. In addition UMLsec can be applied to specify a protocol for the transmission of web application data.

### 3.1.2 *SecureUML*

SecureUML [30] is a UML profile that enables the developer to model Role-Based Access Control (RBAC) as described in section 2.5. The SecureUML metamodel is depicted in figure 3.2, additional authorization constraints can be expressed with the Object Constraint Language (OCL). A *SecureUML dialect* has to be defined to connect a *system design modeling language* as e.g. *ComponentUML* with the SecureUML metamodel. This is necessary, because the possible `Actions` on the predefined `Resources` have to be specified.

*"Similar to SecureUML, UMLsec can be considered as a notation to specify and design secure software systems rather than a security requirements engineering method" [11]*

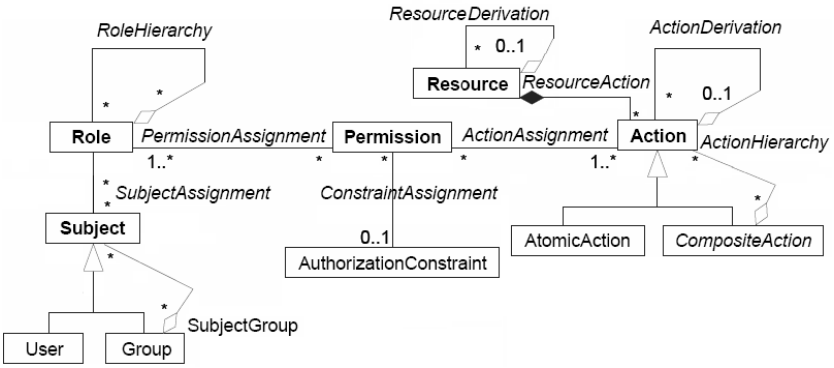


Figure 3.2: SecureUML metamodel (adapted from [9, page 13])

*ComponentUML* is a modeling language similar to a simplified UML class model containing only entities (instead of classes) with methods and attributes. Entities can be connected with associations. An example for a SecureUML dialect which restricts access on methods would be modeled in the following way: `EntityMethod` (ComponentUML) is inherited from `Resource` and provides the `AtomicAction` execution that is expressed by a usage arrow labeled with 'execute'.

With these preconditions, a simple RBAC pattern can be applied to a fictively class `CommentManager` that handles blog comments. Registered users are allowed to write comments, everyone else – «Role» (anonymous) `Users` – has read-only access. Views enable the web engineer to model the weblog application itself independently of the security aspects. They are just added in an extra SecureUML model, as depicted in figure 3.3.

In our approach, we specify role-based execution rights to methods in a *Basic Rights Model* (section 4.4.2) using dependencies instead of the SecureUML association classes, because they are rather confusing due to their copies of method names with an access related return type. Our models are constructed deliberately in a way that they can easily be transformed into a SecureUML representation, as described in section 4.4.3.

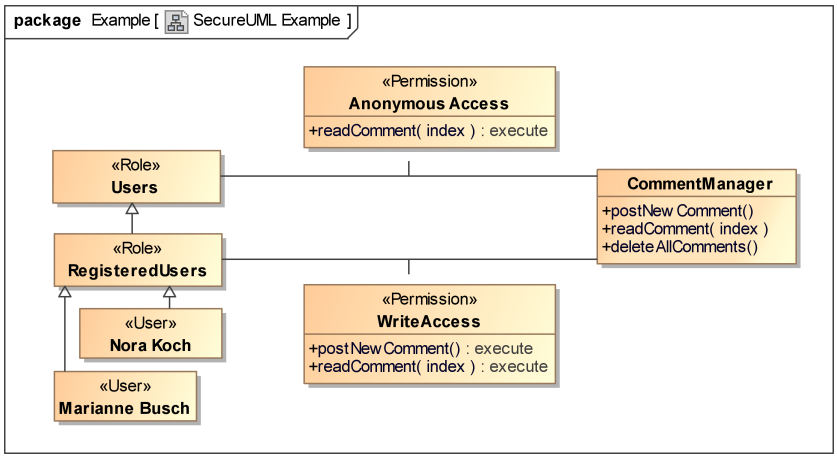


Figure 3.3: SecureUML example

### 3.1.3 *Security modeling for SOA*

Service-Oriented Architectures (SOAs) technically consist of different aspects of orchestrated services. 'Orchestrated' is a buzzword that "describes the automated arrangement, coordination, and management"[4] of services. Security modeling for SOAs focus on the communication between several services in order to make sure it is as secure as intended.

The difference to our approach is that we focus on the model of the web application itself and hence on the details of the implementation rather than on the communication with the outside world. Nevertheless, almost every web application can be extended with an Application Programming Interface (API) to be assimilable in a SOA environment.

Lift for example provides a RESTful API. [7, chapter 15: web services] Representational State Transfer (REST) is an http-based architectural style that describes a uniform interface to resources, e.g. data values, images, etc.

#### 3.1.3.1 *SECTET Framework*

In their book "Security engineering for Service-Oriented Architectures", the authors define security as

> "the sum of all techniques, methods, procedures and activities employed to maintain an ideal state specified through a set of rules of what is authorized and what is not in a heterogeneous, decentralized and interconnected computing system." [18, page 28]

4 Wikipedia: Orchestration. http://en.wikipedia.org/wiki/Orchestration_%28computers%29, last visited 2010-09-10

The thought of having "a set of rules" leads to security policies, which can be modeled later on. They concentrate on four basic security policies:

CONFIDENTIALITY POLICY to ensure confidentiality of the exchanged documents.

INTEGRITY POLICY to specify who is allowed to alter information.

AVAILABILITY POLICY is intentionally put on the same level as non-repudiation. [18, page 32] Usually, Availability means that something e.g. a service or a document is available when needed. The preceding three policies are often referred to as *CIA triad*, according to the first letter of each term.

POLICY MODELS. Discretionary Access Control (DAC) is mentioned as well as Mandatory Access Control (MAC), and RBAC.

In addition, advanced Security Policies include more complex issues like e.g.:

DYNAMIC ACCESS CONTROL POLICIES grant access according to a changing environment e.g. nurses can only access the information of patients of their ward. UCON$_{ABC}$ [40] is a model for usage control, supporting *continuity of access decision* (which means that the access decision is verified during the access so that it is possible to revoke permissions whenever necessary) and *mutability of attributes* (which ensures that subject or object attributes are adapted, which can lead to a change in access decisions). Examples are conflicts of interests that can be resolved by denying access to certain objects.

4-EYES-PRINCIPLE means that access is only granted if two subjects (usually humans) are available at the same time. An example is the access to the data stored on electronic health insurance cards.

Technically this is tackled by reading the e-health card of the patient and the doctor's Health Professional Card at the same time and credentials like passwords can be additionally required. Stereotypes (e.g. «fourEyesPrinciple») and OCL constraints are used in a UML class diagram to determine, which policy is applied. [19, figure 3]

DELEGATION of rights policies allow users to delegate rights to other objects.

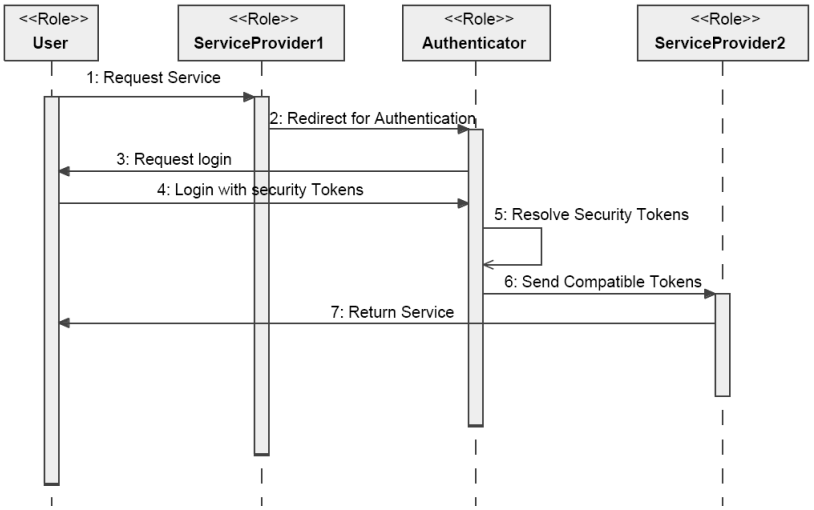


Figure 3.4: Single Sign-on authentication [31, page 6]

BGP (Break-Glass Policy) grants access in case of emergency to e.g. medical data of vital importance.

In order to inspect the communication between the services, sequence diagrams are used as patterns. An example is the SSO authentication as depicted in figure 3.4. It is certainly not satisfactory to model this diagram every time a developer want to use SSO, but this encapsulation provides not only a coherent specification of what is meant by SSO, but also eases the adaptation process, if something has to be slightly changed for a particular system.

#### 3.1.3.2 *UML4SOA*

UML4SOA was developed in the SENSORIA EU project[5], which was coordinated by the LMU from 2005 to 2010. The UML-based modeling language UML4SOA supports constructs such as event handling and message passing.

Security was not in the focus of this project; nevertheless it discussed the inclusion of security aspects as non-functional properties of services. The non-functional metamodel (UML4SOA-NFP, depicted in figure 3.5) provides a monitored `NFContract` as a part of a `ServiceInterface`. This contract specifies non-functional aspects (`NFCharacteristic` e.g. performance, security) that are described in more detail with `NFDimension` elements. Encryption and digital signatures are examples mentioned in a case study of SENSORIA. [13, chapter 4.2] In this way, the developer can model e.g. properties of encryption to specify which part of the message should be encrypted, which algorithm will be used and decide if a signature is necessary. These `RunTimeValues` are controlled by

5 SENSORIA. http://www.sensoria-ist.eu/, last visited 2010-10-20

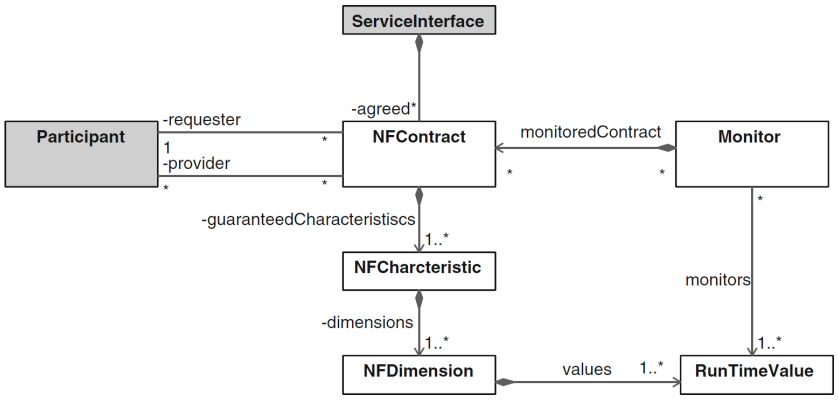


Figure 3.5: UML4SOA. Metamodel of non-functional extensions. [13, figure 6]

the monitor to make sure that the previously agreed contract is fulfilled.

### 3.1.3.3 *SecureSOA*

SecureSOA [32] is a relatively new security language for Service-Oriented Architectures integrated in Fundamental Modelling Concept (FMC) Block Diagrams. FMC is a semi-formal graphical notation to describe software systems. SecureSOA can also be combined with other languages, such as Business Process Modeling Notation (BPMN).

SecureSOA's base (meta) model is shown in figure 3.6, it contains an `Object` (e.g. a web service or a service client), consisting of a set of `Attributes`. The `Interaction` the object participates in is performed on a `Medium` (network) in order to exchange `Information`.

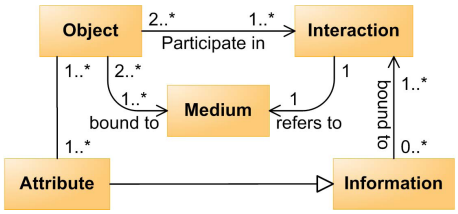


Figure 3.6: SecureSOA. Base model [32, figure 2]

Several security patterns can be modeled with SecureSOA: digital identities for the web services as well as for users, general security policies and authentication and confidentiality constraints. The metamodel of the confidentiality constraint is depicted in figure 3.7. It is noticeable that the metamodel is not directly based on the base model of figure 3.6.

A model example would instantiate e.g. the `Confidential Algorithm Type` with a concrete cryptographic protocol, `Policy Owner` would be set to the name of the service and `Credential Type` would be a particular public key from a PKI. In our opinion this representation of SecureSOA models is rather confusing and the extensive usage of connection arrows in figure 3.7 give rise to this impression.

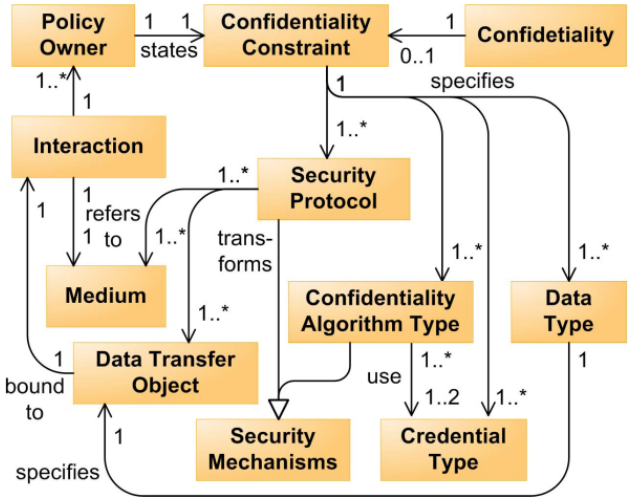


Figure 3.7: SecureSOA. Confidentiality constraint [32, figure 9]

### 3.1.4 *Pattern-based Approaches*

*"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." [49, definition 12]*

Most existing pattern-based approaches are rather at the level of system engineering, than at the level of software engineering. Security patterns are often extended and republished later on. An example is a technical report about secure design patterns [10] which subsumes

ARCHITECTURAL-LEVEL PATTERNS , such as *privilege separation* for reducing the code that runs with more privileges than needed.

DESIGN-LEVEL PATTERNS , like *secure factory*, which returns a particular object after the input of a specific set of credentials.

IMPLEMENTATION-LEVEL PATTERNS , such as *secure logger* to prevent gathering useful information from the logfile of an application.

The report does not deal with privacy aspects. Privacy patterns can be found in the papers of Hafiz [16] and Romanosky et al. [45]. The former describes privacy at the network and transport layer, the latter identifies three online interaction patterns:

INFORMED CONSENT FOR WEB-BASED TRANSACTIONS "how websites can inform users whenever they intend to collect [...] personal data" [45]

MASKED ONLINE TRAFFIC disclose only a very few amount of personal data

MINIMAL INFORMATION ASYMMETRY users should gather information about the services they are using

Many aspects have to be taken into account for each of the patterns and it is useful to keep them in mind while modeling a web application.

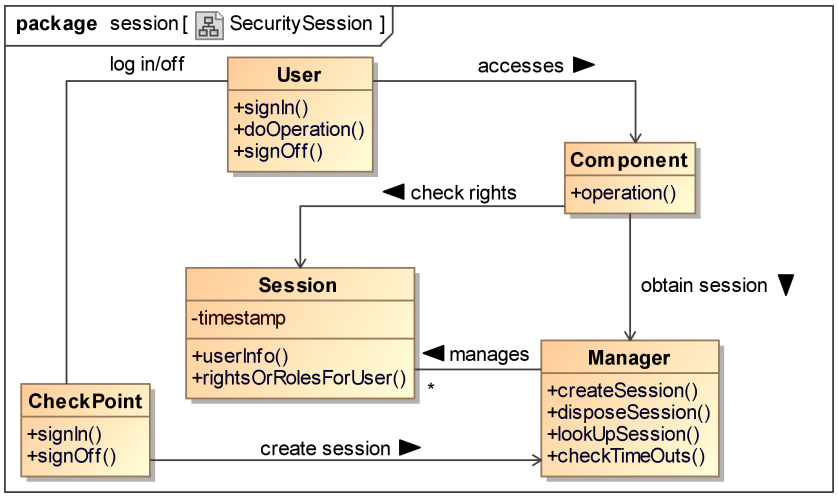


Figure 3.8: Pattern example. Security session (adapted from [50, page 300])

Some of the pattern descriptions contain diagrams, e.g. the pattern of access control with security sessions in [50] (as depicted in figure 3.8). They often provide an UML overview, such as our security aspects in the following chapter. We can see in figure 3.8 that it is important to discuss if we want to model session timeouts and how this can be done in a practical way.

To fulfill the objective of our work, more specific patterns are important, e.g. different login schemata that can be modeled in a more detailed way. This is especially useful, if a standard procedure can be modeled as a pattern (with stereotypes) as well as redefining the pattern in detail. Nevertheless, the developers should pay attention to the general patterns [50, 10, 25], because many aspects of security are out of scope of formal or graphical methods, as described in section 2.10.

## 3.2 Web Engineering

In 1999, Murugesan et al. defined:

> Web engineering is the establishment and use of sound scientific, engineering and management principles and disciplined and systematic approaches to the successful development, deployment and maintenance of high quality web-based systems and applications. (republished in [35])

The need of web engineering methods may even have grown with the increasing complexity of current web applications, because an

up-to-date RIA can quickly become as complex as a usual desktop application.

The following subsection provides an introduction to UWE. Our security extension *UWEsecurity*, as described in chapter 4, is based on this web engineering method, but can also be used together with other UML-based approaches. Subsequently, some existing security aspects of UWE are illustrated and a comparison with the Web Modeling Language (WebML) regarding security issues is drawn. At the end of this chapter, two smaller web security modeling approaches are presented in brief.

### 3.2.1 *UWE*

*"UWE uses 'plain' UML notation and UML diagram types whenever possible for the analysis and design of Web applications, i.e. without extensions of any type" (UWE Homepage)*

The increasing application complexity is another reason for choosing UWE for this work, as UWE is based on UML, which can be extended using the profile mechanism provided by UML. The traditional UWE-Profile defines basically four views for the separation of concerns: Content, Navigation, Presentation and Process. The following sections provide a quick overview of each of these views, as far as needed for the next chapters. Further information, some modeling examples and a tutorial can be found on the UWE website.[6]

#### 3.2.1.1 *Content*

The content model comprises classes which are needed by the future application for domain specific tasks. No UWE stereotypes are defined for those classes at the moment, but of course each model can be enriched with information of other UML profiles, which specify extensions applicable for class diagrams.

#### 3.2.1.2 *Navigation*

Class diagrams are used for modeling the navigational view of a web application as well, but there is an ongoing discussion to use hierarchical state charts instead. This provides additional information as it would be more obvious which parts of a web page can be displayed and changed independently. Furthermore sessions and connections could be represented as discussed in more detail in the next chapter.

UWE stereotypes are provided to distinguish several types of UML classes and associations so far, as can exemplarily be seen in figure 3.9.

An example of a model of an online address book supporting the search, creation and deletion of contacts is depicted in figure 3.10. For nodes and links the stereotypes «navigationClass»

6 UWE. http://uwe.pst.ifi.lmu.de, last visited 2010-12-24

| | | | |
|---|---|---|---|
| ☐ | navigationClass | ▤ | menu |
| ≡ | index | ? | query |
| ⇢ | guidedTour | Σ | processClass |

Figure 3.9: Names and symbols of UWE navigation class stereotypes

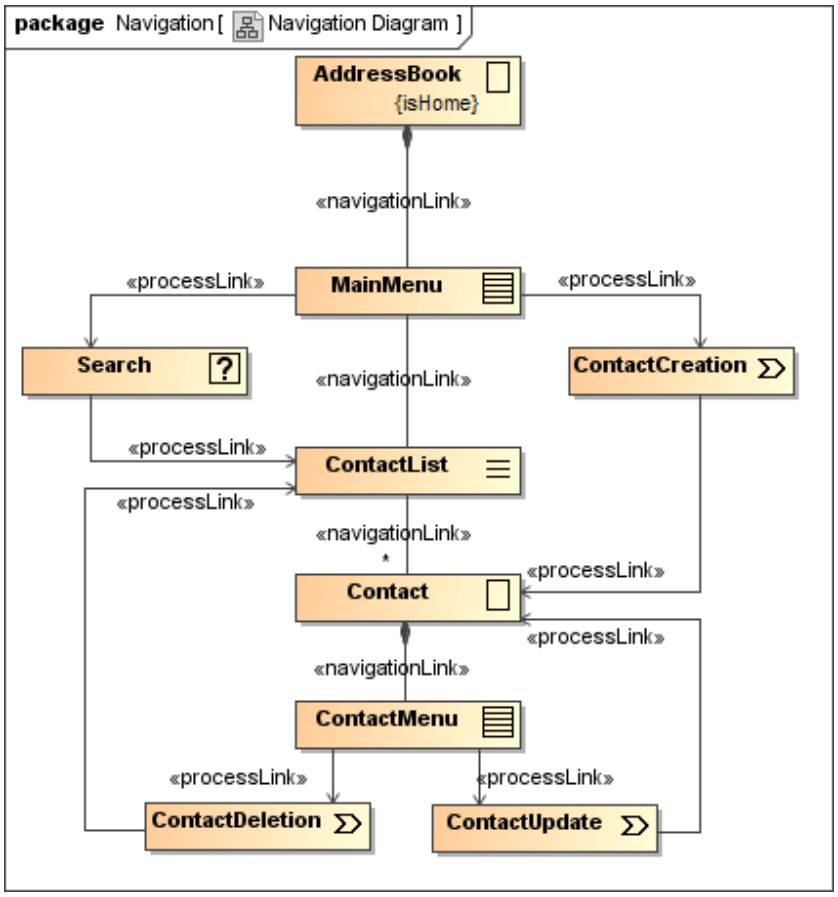


Figure 3.10: UWE example. Navigation class diagram

and «navigationLink» are used and the tagged value {isHome} indicates the starting point of the navigation structure. A «menu» (▤) can be inserted to navigate to *different* classes, as `Search`, `ContactList` and `ContactCreation` in our example. The stereotype «index» (≡) is used to list a number of objects of the *same* type, as miscellaneous `Contact` objects. The content update, i.e. modification of contact data, deleting a contact or adding a new contact, is modeled with so-called process nodes and integrated in the navigation flow by process links. A «query» (?) (as can be found on the `Search` class) is a kind of «process class» (Σ), because searching something is a very common task in the web.

We can see that the assignment of navigation nodes to pages is not specified in the navigation model (but to some extent in the presentation model, see section 3.2.1.3). Sometimes only a part of the page changes; for example a RIA implements our model

of figure 3.10 and permits to edit a contact and allows to search for others in the meantime. Other realizations of that model might reconnect to the server for each of the elements shown in the model and it is not clear which navigation node can only be used within a previously established session. These aspects are discussed in further detail in chapter 4.

#### 3.2.1.3 *Presentation*

In order to define the rough structure of the web layout, the presentation view models not only the way elements are arranged, but also allows the engineer to nest them, using UML properties. Furthermore, special stereotypes and tags for RIAs specify the behavior of interactive elements as text fields that provide automatic completion while the user is typing. It is obvious that security affects all views, in this case it would be desirable to define, which element can be displayed and which one should not be visible. [3] This depends on a full-fledged security model, therefore our work first and foremost focuses on the security basics.

#### 3.2.1.4 *Process*

UWE provides two process modeling views: on the one hand the process structure can be defined, showing dependencies between several processes (⧁) in class diagrams. On the other hand, the workflow of the processes itself can be modeled with activity diagrams. UWE stereotypes indicate if something is a user (🏃) or a system action (▯).

#### 3.2.1.5 *Existing Security Engineering Approaches for UWE*

Two aspect-oriented approaches exist for UWE:

*Zhang et al.* First, Zhang et al. proposed an aspect-oriented technique for modeling access control in web applications, based on the navigation model of UWE. [54] State machines are used to specify the process of granting access to navigation nodes. The advantage is that access control rules can define nodes (e.g. web pages like 'LogOn' or 'Error') to be displayed, instead of just deciding if a user has access or not, as depicted in figure 3.11. To connect one or more aspects with a navigation node, aspects are modeled as (nested) containers wherein the nodes are included. Our approach also uses state machines, but integrates RBAC directly in a new UWE navigation model, as described in further detail in section 4.2.

*Pramod et al.* Second, Pramod [42] sketched the same authentication aspect (without quoting Zhangs work) and added a login tracer to deal with multiple unsuccessful login attempts. The aim is to execute several aspects in a row and to describe each of them with a

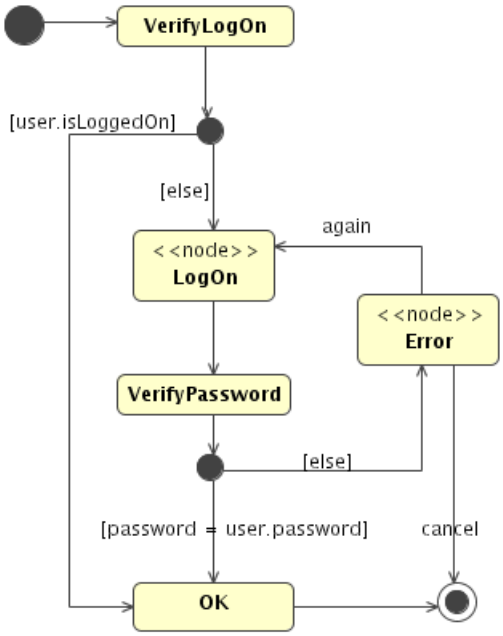


Figure 3.11: State machine specifying an access control rule [54]

state chart. Every aspect is attached to a component using UML component diagrams, but the relation to the aspects in UWE models is not clearly shown.

A subsequently published paper [43] is closer related to the implementation of security aspects in java. Examples are cross site scripting, sql injection, authentication, authorization and session hijacking attacks. Unfortunately the paper is not very detailed with respect to the modeling of the proposed aspects.

### 3.2.2 *WebML*

The Web Modeling Language (WebML)[7] is another web engineering method with a focus on data-intensive web applications. Originally, a particular graphic notation has been invented for WebML, e.g. for hypertext elements as login units, data units or several units to modify database objects. In the meantime, WebML can also be expressed using a WebML-UML profile. [33] Nevertheless, Entity-relationship models (ERMs) are still in use and hence not fully replaced by UML.

As in UWE, WebML defines different views. For example a content view for data organization, a composition view to define pages and their subpages, a navigation view and a presentation

7 WebML Homepage. http://www.webml.org, last visited 2010-10-16

view. A good introduction can be found at the homepage of Brambilla.[8]

Security issues modeled by WebML are access control and site view assignment, which means a group of users has access to a set of sites. The dependency of users and groups are intended to be saved in a database and login and logout units provide a basic session control by storing the current Object Identifier (OID) of the logged in user and the associated group as predefined global parameters. WebRatio[9], a CASE tool for BPMN and WebML can be used to model a business process for each role a user can take on and generates a basic web application comprising user authentication, role assignment and authorization to tasks. If users are not entitled to do a task, they cannot even see it in their task list. In this way, too many error messages are avoided.

### 3.2.3 *Other Approaches*

*Oberortner et al.*

There are some other rather isolated approaches for the integration of security and web engineering. Oberortner et al. [36] combined pageflow in the web with RBAC. The paper is motivated by Model-Driven Software Development (MDSD), but the resulting example is a very simple one, even though the related model is rather big and unintuitive. That is because the metamodel provides a flexible and therefore complex, decision based system for the navigation, which comes along with the `If`, `ElseIf`, `Else` classes, depicted in the metamodel (figure 3.12). Apart from this, they present the navigation structure as simple state machines for each role, comparable to WebML's BPMN diagrams. Our approach is to use complex states that integrate different roles in the navigation structure. This makes the reuse of nested state diagrams possible.

*Aedo et al.*

An older approach from Aedo et al. [1] uses the *Ariadne Development Method for Hypermedia*. The paper focuses on authorization and authorization propagation mechanisms, i.e. the inheritance of permissions, which are associated with subjects in a RBAC manner. Grouping of objects is also taken into account. Graphical modeling techniques are mentioned, such as a structural diagram showing the structure of objects, but the access control itself is mainly based on Access Control Lists (ACLs). Security access categories are *browsing*, *personalizing* (creating personal entries) and *editing* (the general structure of the page). Compared to everyday social web communities as facebook, these categories may be useful especially in a preliminary stage of modeling. In

8 WebML. Tutorial with audio and slides. `http://dbgroup.como.polimi.it/brambilla/webml`, last visited 2010-10-16

9 WebRatio. `http://www.webratio.com`, last visited 2010-10-15

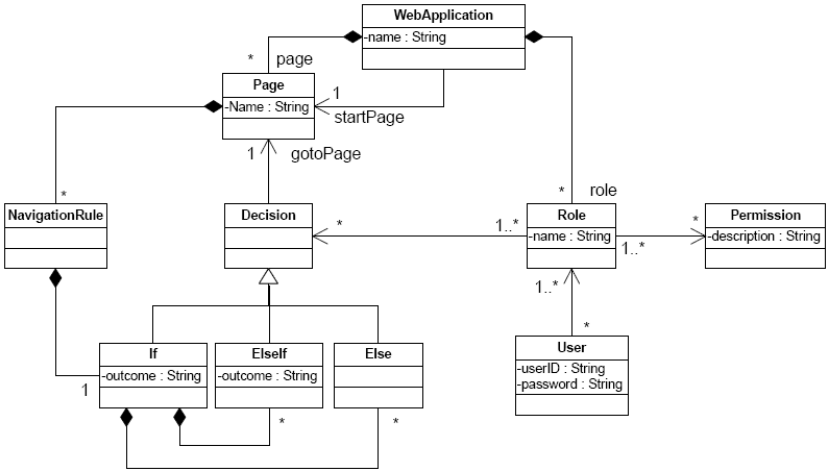


Figure 3.12: Pageflow and RBAC metamodel [36]

our work, we focus on RBAC, because it is independent of its field of application.

# Part II

# UWESECURITY

# 4

# SECURITY ENGINEERING FOR WEB APPLICATIONS

As mentioned in chapter 3, our approach called *UWEsecurity* can be combined with UWE, but *UWEsecurity* can also be used independently or be added to other UML profiles. This chapter introduces *UWEsecurity*: Section 4.1 presents a way of modeling the requirements of secure web applications. In section 4.2, the navigation state model is specified, which also is used in section 4.3 for common security patterns that can easily be included as substates in the navigation state diagrams. Finally, section 4.4 defines the *UWEsecurity* "basic rights" Role-Based Access Control model, which is transformable in a SecureUML model.

*overview of UWE models*

Before describing the features of *UWEsecurity*, an overview of the purpose of all resulting UWE models is given (cf. figure 4.1). Further information to the traditional UWE models can be found in section 3.2.1.

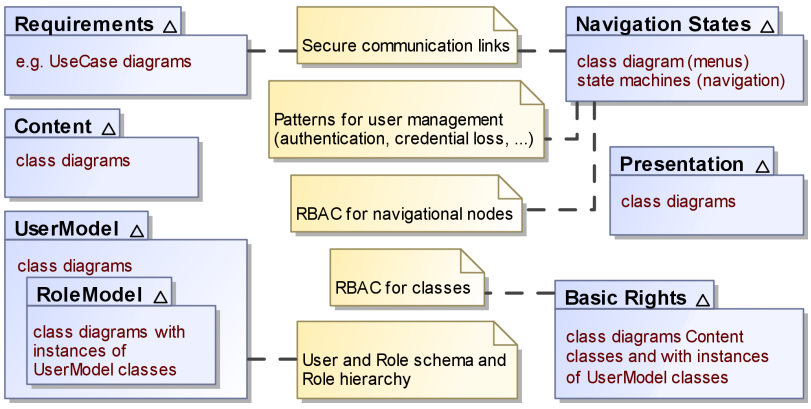


Figure 4.1: UWE with UWEsecurity. Model overview

CONTENT. The content model is used to represent relevant concepts of the domain and their relationships. In UWE we use a UML class diagram for the visualization of the content model.

USER MODEL. A user or context model can be used to collect information needed for adaptation (which is not used in the following examples). Furthermore, it is useful to add characteristics of users and roles, as kinds of `User` and `Role` classes to work with *UWEsecurity*. Of course their relation to content classes can be depicted in content diagrams. The

user model can contain a *UWEsecurity role model* that represents the role hierarchy, e.g. with a snapshot of existing role and user instances and their relations, usually illustrating the state after the installation (see section 4.4).

BASIC RIGHTS. The basic rights model specifies access control. For that it connects the role model with classes from content using dependencies with *UWEsecurity* UML stereotypes and UML tags.

NAVIGATION. UWE's navigation model is used to represent navigable nodes of the hypertext structure. Links between nodes stand for the possibilities of navigation the user has.

The *Navigation Classes Model* is the classic navigation in UWE, which is much easier to use than the *UWEsecurity* navigation states model, but also less expressive. UML class diagrams are used for representing the navigation nodes.

Section 4.2 introduces the *UWEsecurity navigation states model* that includes security aspects like secure connections, sessions and access control on navigational nodes. In addition to the UML state machines, the structure of navigation menus can be represented with class diagrams.

PRESENTATION describes not only the rough layout of the pages, but also RIA features as autocompletion in search fields. Again, class diagrams are used that are extended with UWE stereotypes.

PROCESS is divided into two views: On the one hand, the process structure model describes the relations between the different processes, which are related to the classic navigation. On the other hand the process flow model comprises activities for the processes.

REQUIREMENTS. The requirements analysis is supported by UWE by use case diagrams and activity diagrams, as described in the following section.

## 4.1 Requirements Analysis

Use cases and roles in use case diagrams are one part of modeling the requirements of a web application. Another part is made up of activity diagrams with special stereotypes, but in this work we did not make use of that UWE feature. [27]

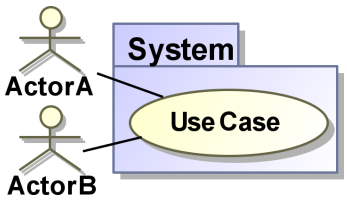


Figure 4.2: Use case example

Before describing our extensions for use case diagrams, we have to

*associations in use case diagrams*

specify how to read them: the association between two actors and one use case (see figure 4.2) can have two meanings: on the one hand it can be seen as 4-eyes-principle (two actors are needed for one use case), on the other hand, it expresses that both actors can interact with the use case individually. [21, p. 184] The latter case is the one, which is used in this work.

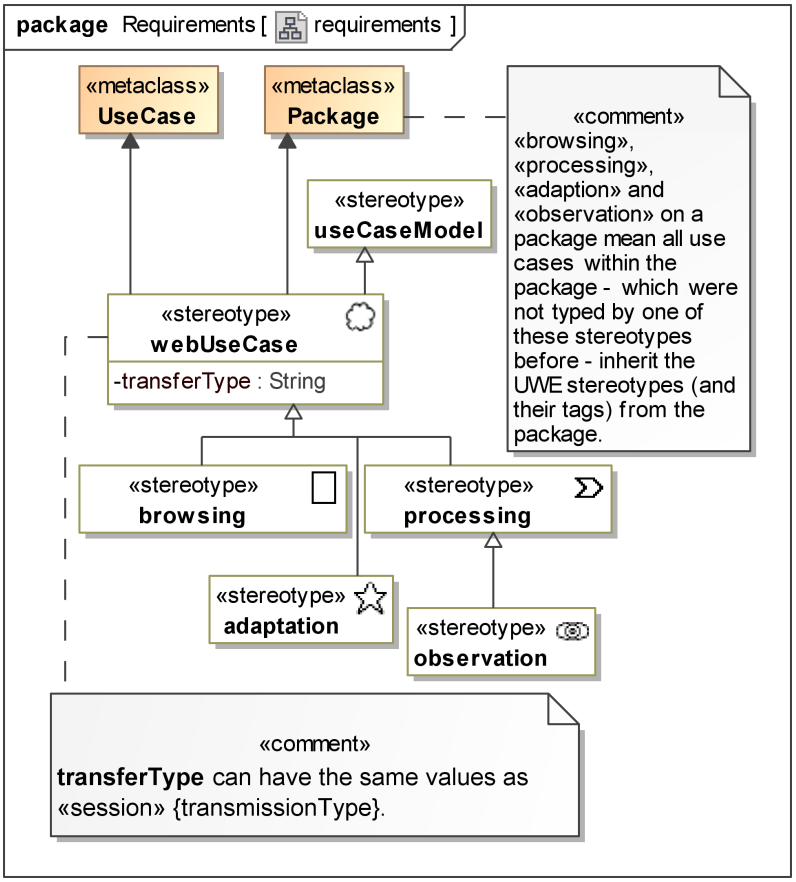


Figure 4.3: Part of the requirements profile

*UWEsecurity extension of the UWE requirements profile*

Figure 4.3 depicts a part of the UWE profile for requirements. An additional metamodel is not necessary for *UWEsecurity*, as all required information can already be found in the profile. The stereotypes «browsing» and «processing» and their descendants are from [27], but they had been renamed (from «navigation» and «process»), because the stereotype «process» exists in the UML profile with a different semantics. In RIAs it is not always clear whether a use case only refers to browsing activities without processing something, as e.g. a generated list from a "list all" operation is processed in a way, too. Therefore, we suggest to use «processing» in case some direct input is given (like a search string) or in case of a significant database modification (like creating a new user).

The UWE profile is extended with the rule which is described in the upper comment in figure 4.3, because previously the men-

tioned stereotypes could only be added to use cases, not to packages. The only new stereotype is «webUseCase» ( ); it denotes not only the 'web' character of a use case, but also makes it possible to specify a {transmissionType} i.e. a mode of transmission for transferred data. Usually, no technology is specified – as the detailed realization is determined later on. For this reason in our examples we use the first letters of the underlying concepts (`c` for confidentiality, `i` for integrity and `f` for freshness).

## 4.2 Navigation State Model

The traditional navigation class model does not specify the navigation flows exactly, because it is not defined when the user can go back or access several nodes in parallel. Hence, the navigation class model is not powerful enough to include basic security issues as a role management (if some users are allowed to access different nodes than others) or secure connections.

*the diagram type of a new navigation model*

Therefore, we need another, more precise navigation model for *UWEsecurity* to represent not only navigation, but also security features. We selected UML in order to describe state configurations, where it is possible that more than one state is active at the same time. This is needed when the user has more than one part of the navigation that switches independently to a state in which other links are offered. Activity diagrams were also taken into consideration, but they are too fine-grained, because the navigation view should concentrate on navigation features and not on single actions. Furthermore, parallel actions cannot be specified as precisely as in state machines. If the communication between the server and the client necessarily is to be modeled in detail, activity or sequence diagrams can be used additionally for rather small parts of the system.

The navigation state model can be used side by side with the UWE navigation class model, as can be observed in our case studies in sections 6.2.4 and 6.2.5. The traditional navigation class model represents only the structure of the navigation and the navigation states model additionally allows the modeler to specify the behavior of the navigation, i.e. when a node can be activated and when not etc.

*the navigation state model profile*

Figure 4.4 presents the profile of the navigation state model. A «navigationMenu» ( ) is a UML class with operations that specify the functionality of navigation menus for web applications. Several classes can be abstract and of course inheritance can be used. In the containment tree, the navigation state machines are stored as descendants of these «navigationMenu» classes. The other stereotypes and tags that are shown in figure 4.4, are introduced in the following subsection.

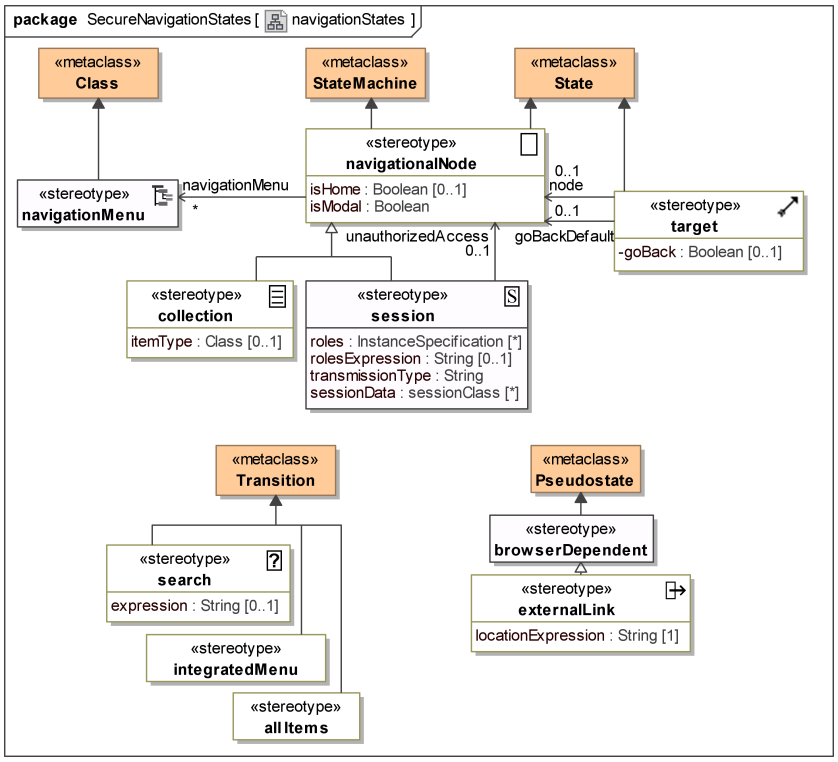


Figure 4.4: Navigation states profile

### 4.2.1 *Web Navigation with State Machines*

A convenient navigation state model to represent the navigation in a web application is specified in the following. Every stereotype and all tags have to be specified exactly. The following pattern is used for the specification:

- *Stereotype* name(s) and tag(s)
- *Purpose* (directly following the name)
- *Constraint* – if necessary
- *Definition* of the transformation for replacing the stereotype-/tag with a longer version without this stereotype, if applicable. The transformation from a version with *UWEsecurity* stereotypes to a plain UML version needs at most two steps.
- *Abstract example* for complex definitions, especially those that require transformations. Abstract means that the naming of the definitions is used instead of real world labels. Further examples can be found in our case studies in chapter 6 and appendix A.

#### 4.2.1.1 *Navigational Nodes*

A «navigationalNode» ( ) is the basic state or state machine stereotype in our navigational state model. 'Navigational' refers

to the view and the granularity of the state machines, because states that do not change the navigational behavior are represented as one state.

SPECIFICATION 1

**«navigationalNode» ( )**

*«navigationalNode» ( )*

Some states (or state machine configurations, i.e. a set of active states in orthogonal superstates) can be accessed directly. This is necessary for explicit access of a special state via a URL. Usually the assignment between a navigation state configuration and URLs is not modeled explicitly in order to keep the diagrams comprehensible. Nevertheless it is possible to represent this behavior by adding a choice between the initial node and the {isHome} state and by using a new variable in the model called 'location'. With guards, effects and choices, the location variable can not only be helpful for direct access, but also for changing the location path according to the current state of the web application.

SPECIFICATION 2

**«navigationalNode» ( ) {isHome}** specifies that this is the main node ($\widehat{=}$ the first state) the users go to when accessing the web application. Usually it is related to the home page of the whole application.

*Constraint.* Exactly one «navigationalNode» has the {isHome = true} tag within a web application.

SPECIFICATION 3

**Default stereotype «navigationalNode» ( ) within other «navigationalNode» states**

Per default, every state which is nested within a state denoted by the stereotype «navigationalNode» (or inherited stereotypes) is assumed to represent a navigational node. State machines can contain outermost states without stereotypes, if they are always used as submachine states in a valid context, i.e. a «navigationalNode» stereotype is set to the submachine state or to parental states of the submachine state. This avoids overfull state diagrams, because stereotypes only have to be set when they represent new information, such as a substate, which redefines the {unauthorizedAccess} behavior or a state that is the target «navigationalNode» of the {unauthorizedAccess} tag.

SPECIFICATION 4

**«navigationalNode» ( ) {isModal=true}** guarantees that only this state is active and that it can only be left by a transition that goes back to the previously active state. All parallel regions are restored after having left the modal state. This represents the navigational behavior of a modal dialog. An example is depicted in figure 4.5; the main advantage is that this abbreviation can be used within state machines without taking care of the nesting.

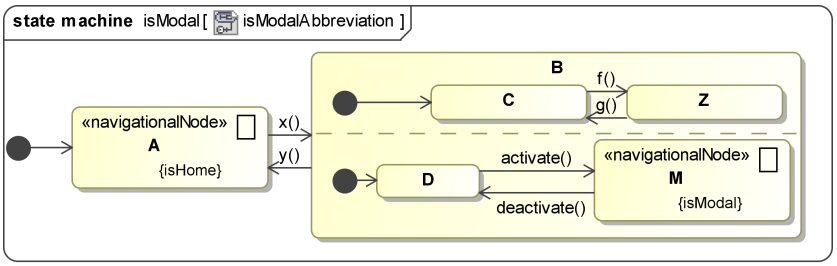


Figure 4.5: «navigationalNode»: {isModal} abbreviation

*Constraints:*

1. Exactly one transition targets the modal state with a unique action., i.e. an action that cannot occur elsewhere at the same time.
2. Exactly one transition makes it possible to go back from the modal state to the previous state.
3. These two transitions are the only ones that are connected with the modal state.

*Definition.* The transformation consists of the following steps: A new composite state $z$ is added that includes the outermost state machine of the web application, i.e. the state with the tag {isHome = true} and connected states. A new initial state is added, which is connected with $z$. The modal state $m$ is moved outside of $z$ and the transition that comes from $m$ targets a new history state in $z$, instead of the state before $m$. For transitions that cross state machine borders, entry and exit points are used.

*example for «navigationalNode» ( ☐ ) {isModal=true}*

Figure 4.5 depicts an example in which the abbreviation with the «isModal» stereotype is used. If states in both regions of `B` are active, `M` can be activated and inhibits all other transitions, in this case `f()`, `g()` and `y()`. The extended version[1] in figure 4.6 shows the outsourced state `M` and the new deep history state within the outermost state `Z*`. The drawback is that entry / exit actions have to be used with caution when modeling with the «isModal» abbreviation, as it is not obvious that they are executed when entering / leaving the modal state. If the modal dialog is displayed within a secure session, the state machines are not quite correct, because we still assume that the modal dialog is implemented by using the given connection.

#### 4.2.1.2 *Sessions*

Sessions are a common concept in the world of the web. They allow the users to navigate in the web application without losing

1 States in extended versions of the examples are renamed from 'State' to 'State*' in order to emphasize their relation

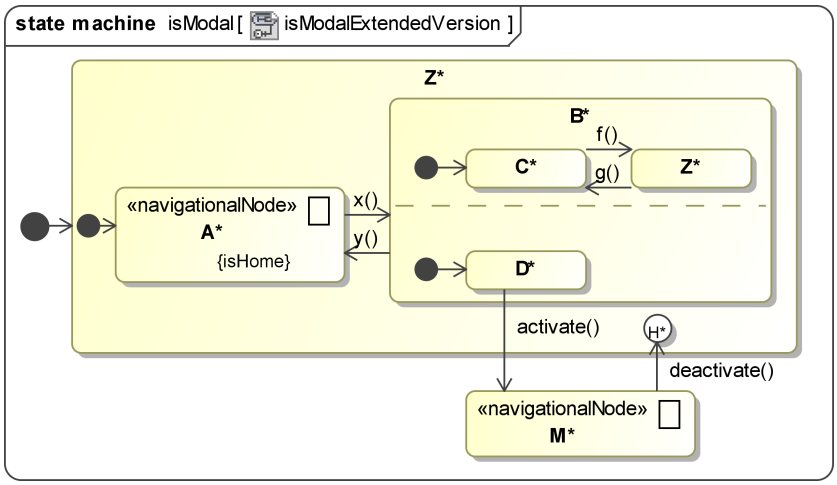


Figure 4.6: «navigationalNode»: {isModal} extended version

the context, like authentication information or the items they added to their shopping cart. Often, sessions are combined with authorization and it is important how long a session is active, e.g. an online bank closes the session after a few minutes of inactivity and a personalized TV guide allows the users to return a few weeks later without an explicit authentication. Furthermore, sessions allow to specify the type of transmission, which can be independent of the user-related sessions.

As can be seen in figure 4.4, the «session» stereotype inherits the «navigationalNode» meaning that a session *is* a navigational node, which can be a state or a state machine. Some useful features are not depicted in the profile, for example there is no need for defining an explicit logout mechanism, because state machines support this out of the box by adding a transition with the trigger `logout()` or even a time-referenced action, as modeled in detail in figure 6.7 (section 6.2.4).

SPECIFICATION 5

*session ( S )*

**«session» ( S ) {sessionData}** allows the modeler to explicitly specify session classes («sessionClass») that are available during the session. It can be used to store session variables as known from many web frameworks.

Please notice that the former «visitClass» introduced in [28, p. 25] is renamed to «sessionClass». Before, it had been rarely used, probably because of the misleading name and the unfamiliar way to apply it at a session, rather than directly denoting the data that is available during a session.

SPECIFICATION 6

**«session» ( S ) {transmissionType}** specifies the type of data transmission between interacting instances, such as the client and the server. The value can e.g. be set to 'cif' for confidentiality,

integrity and freshness. (May later be implemented as SSL/TLS connection.)

SPECIFICATION 7

**«session» ( 🅂 ) {rolesExpression}, {roles}**

The permissions of a session can be set by the following tags:

- *{rolesExpression}* – To access a node, the `rolesExpression` has to be true. Usually, '`currentUser`' refers to the currently active user instance.

- *{roles}* – 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. If both tags are set, they are combined according to the following definition.

*Constraint.* The user model contains a `User` class and `Role` class or enumeration. Usually, a `User` class is connected with a `Role` (multiplicity *, role name '`roles`') as presumed for this definition.

*Definition for RBAC (Role-Based Access Control).*
{rolesExpression} and {roles} are combined in the following way:
`rolesExpression` := `R` & `rolesExpression`, where
`R` = `(currentUser.roles->includes(`$r_1$`) |...|`
`currentUser.roles->includes(`$r_n$`))`
($r_1, \ldots, r_n$ are the role values for {roles}).

The {roles} tag is deleted at the end of the transformation and the final {rolesExpression} tag also is used in the definition of {unauthorizedAccess} (see specification 8 in the next paragraph).

SPECIFICATION 8

**«session» ( 🅂 ) {unauthorizedAccess=navigationalNode}**

refers to a node, users without the appropriate permissions (see specification 7) should be redirected to. An example is depicted in figure 4.7, but of course the tag is more useful if the target state is located in other state machines or if the target is often redefined within state machines.

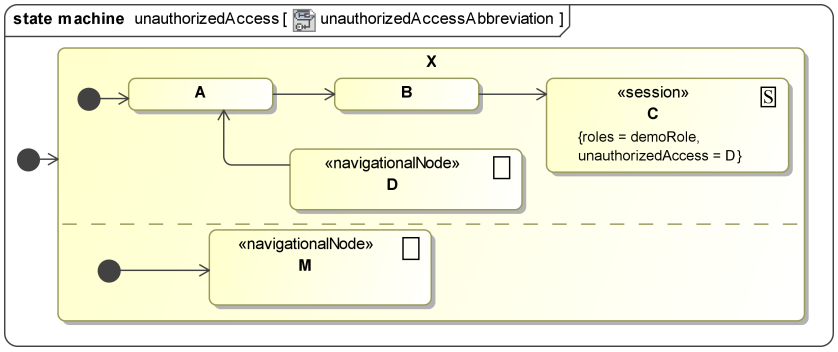


Figure 4.7: «session»: {unauthorizedAccess} abbreviation

*Definition of {unauthorizedAccess=s} on «session» c (s is typed by «navigationalNode»).* c is wrapped in a new composite state z. All transitions that started/ended at c, now start/end at z. A new initial node and a choice with the final {rolesExpression} constraint (see specification of {roles} and {rolesExpression}) is added to z and connected by a transition. The choice has two outgoing transitions: One with the guard [true], leading to c and one with the guard [false], targeting s. Targeting s should also be possible if s is located in another state machine, therefore the abbreviation with the stereotype «target» and the tag {node=s} is used as intermediate step to bypass the borders of the state machines (see the following subchapter).

*example for «session» ( 🅂 ) {unauthorizedAccess}*

An example for the usage of the tag {unauthorizedAccess} is depicted in figure 4.7. Session `C` is only available for users with the role `demoRole`. If a user is not assigned to this role, the access to node `C` is denied and he is redirected to the node `D`, as specified in the {unauthorizedAccess} tag.

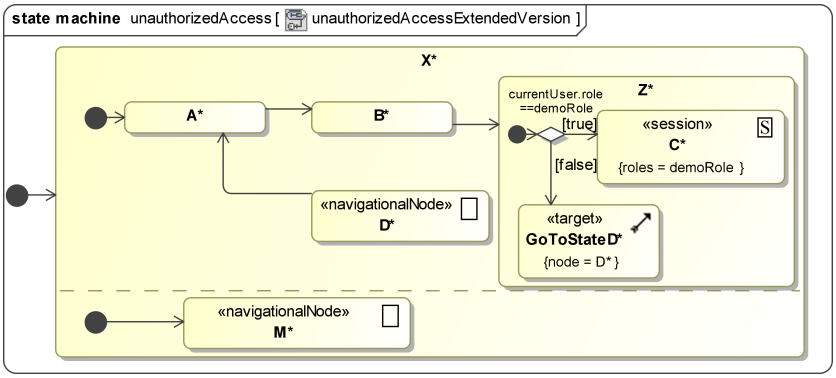


Figure 4.8: «session»: {unauthorizedAccess} extended version

Figure 4.8 shows the extended version with a new state `Z*` and a choice which tests the assigned role. The new state `GoToStateD*` is typed by «target» with the tag {node=D*}, which in this case means that the transition with the guard [false] ends at `D*` and the state `GoToStateD*` is deleted. As a consequence, the process of replacing the *UWEsecurity* abbreviated types with the extended versions remains linear (in relation to all states). At most two steps are required: firstly, the replacement of «session» {unauthorizedAccess} and afterwards the one of «target» {node}.

#### 4.2.1.3 *Targets*

*«target» ( ➶ )*

The «target» stereotype ( ➶ ) is needed in case a web application directly implements the functionality of going back to the previously displayed page. Another field of application is the "go to {node}" option. It is not recommended to use it directly, because diagrams become very confusing, much the same as early BASIC

programs that used 'goTo' instead of expressing directly that a loop or a method call is meant. Nevertheless, it can be very useful for defining the aspect of unauthorized access behavior (see previous subsection) without flooding the diagram with transitions and entry or exit points.

SPECIFICATION 9

**«target» (↗) {goBack}, {goBackDefault} and {node}** that can be set on a UML state allow the modeler to use a shortcut for:

- either going back to the previous state, which is defined as the state before the state that has a transition to the state with the {goBack} tag. (see figure 4.11)

  (The {goBackDefault} tag specifies an alternative node, in case the state is accessed directly and not from previous states.)

- or going to another {node} (see figure 4.9).

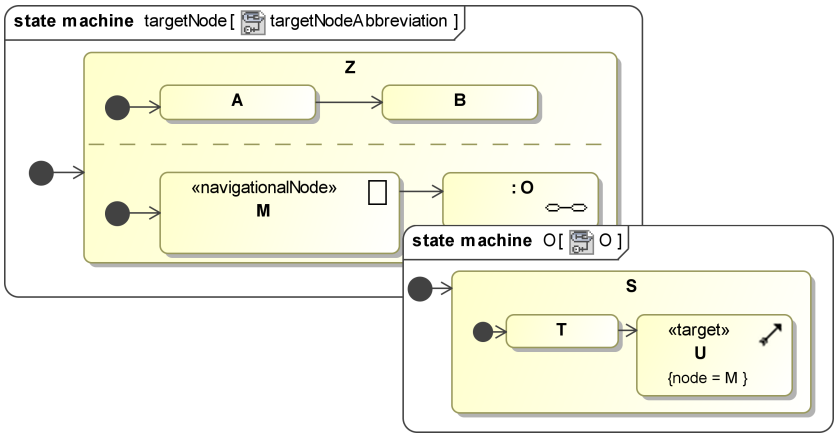


Figure 4.9: «target»: {node} abbreviation

*Constraints:*

1. Only one transition can lead to a state stereotyped by «target». If there are more transitions, the state typed by «target» has to be copied before the definition of the transformation can be applied.

2. If the tag {goBack} or the tag {goBackDefault} are set, the tag {node} cannot be set at the same state. If only {goBackDefault} is specified, {goBack} is added before the definition is applied.

3. {goBack} only: if there is a case where it is not possible to go back (because no variable x is set, cf. definition), the tag {goBackDefault} has to be defined.

*Definition of {goBack}.* Let s be the state before the «target» state r. t is the transition from s to r. Let x be a variable, which had not been in the set of all variables before.

To all outermost states (no pseudostates) Q that can be left by a transition which leads to s, $exit/x = q'$ is added ($q'$ is a unique state identifier of q and q is a member of Q). The guard $[x == q']$ is added to t. t is copied for each q and the copy of t targets:

CASE 1 a new deep history in a region of q, if q is a composite or orthogonal state[2] deep history states are only defined for composite states (not for orthogonal states), which may result from a inadequate renaming between previous UML versions.

CASE 2 q, else.

A transition can cross state machine borders using entry and exit points. Afterwards r is deleted.

If the variable x can be null, because of any kind of direct access (see specification 1), then the {goBackDefault=p} tag is used. This means the tag {node=p} is set and the tags {goBack} and {goBackDefault} are deleted.

*Definition of {node=p}.* (Should be used rarely, it is mainly defined to handle the aspect of unauthorized access.) Transition(s) and exit/entry points – if required – are added to connect the state before the «target» state with p. At the end of the transformation the state typed by «target» is deleted.

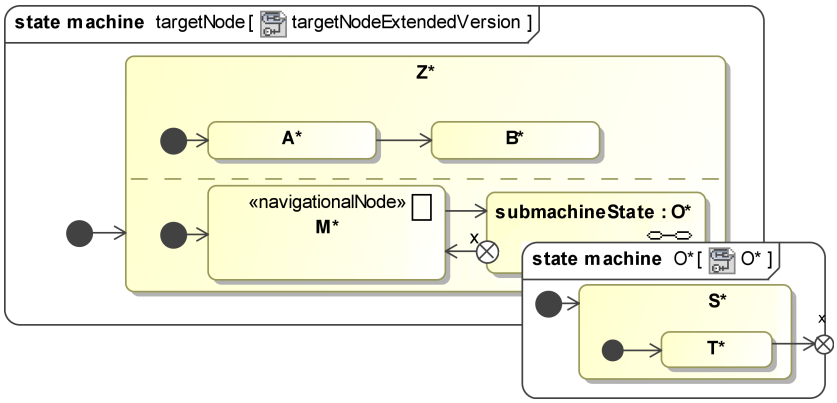


Figure 4.10: «target»: {node} extended version

*example for «target» (➶) {node}*

Figures 4.9 and 4.10 illustrate the definition of {goBack}. Figure 4.9 depicts an example that uses the abbreviation, where the state T that is located within a substate of the state machine O

2 The deep history is expected to recover the previous state of *all* regions of q. In [37, p. 557]

should target the state `M` (of a state machine, which includes `O`) with its outgoing transition.

This behavior is explicitly shown in figure 4.10. For crossing the state machine borders with the transition, a new exit point `x` and a transition from the exit point of the submachine state to the state `M` is added. The transition to `U` now targets `x`. Of course the advantage of «target» is linear the number of nested state machines.

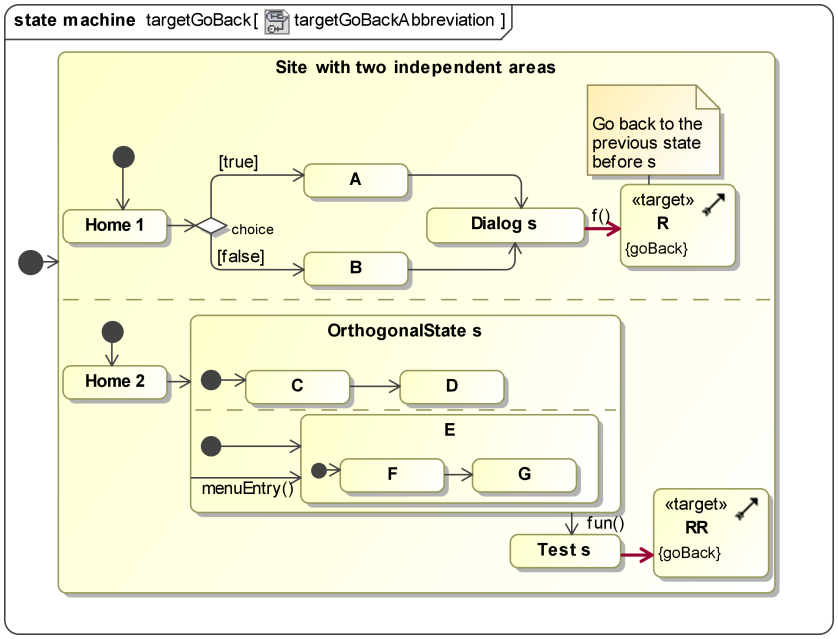


Figure 4.11: «target»: {goBack} abbreviation

*example for «target» (↗) {goBack}*

An example for the application of the «target» {goBack} definition is presented in figure 4.11. In the first region of the outermost state, the red, bold transition is expected to navigate to the state that had been active before `Dialog s` (expressed with the state `R`). In the second region, the difference is that the previous state is an orthogonal state with substates.

In figure 4.12 the extended versions are depicted. In the upper region, two red, bold transitions are used with guards, checking in their condition a variable, which is set in the exit actions of the states `A*` and `B*`. In the lower region, another variable is set and because one of the previous states (the only one in this case) is an orthogonal state, a deep history is used. Please remember our assumption for deep history states: they are expected to recover the previous state of *all* regions of the surrounding orthogonal state.

#### 4.2.1.4 *Collections*

SPECIFICATION 10

**«collection» (🗏) {itemType=ItemClass} and «allItems»**

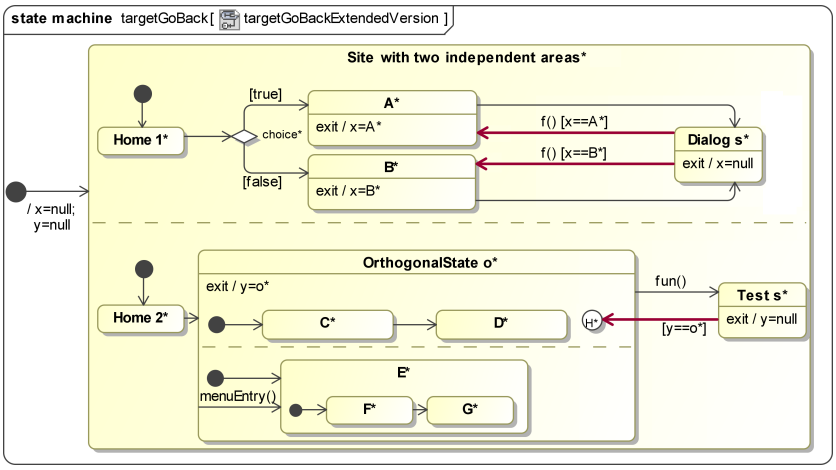


Figure 4.12: «target»: {goBack} extended version

*«collection» ( )*

«collection» is used to simplify the presentation of collections e.g. lists with several items that are typical for the web. The tag {itemType=ItemClass} targets an `ItemClass` (from the content model). An example is depicted in figure 4.13.

The «collection» stereotype is defined on:

- a submachine (state) to denote that it contains one or more states with the «collection» stereotype.
- a state in order to specify the meaning of the outgoing transitions:
  - By default, outgoing transitions are thought to come from a particular item of the collection. Typically, trigger (or effects) on outgoing transitions have a first parameter `i:ItemClass`. To avoid repetitions, an underscore '_' can be used instead of `i:ItemClass`.
  - Outgoing transitions with the stereotype «allItems» are intended to come from the whole collection, not from single elements.

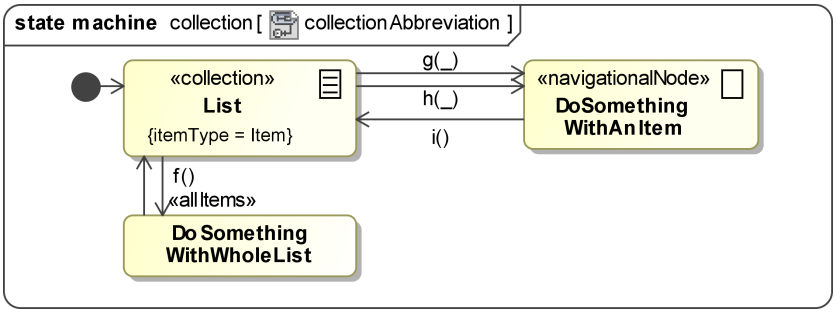


Figure 4.13: «collection» abbreviation

*Constraints:*

1. «allItems» can only occur on an outgoing transition of a state that is stereotyped by «collection».
2. All outgoing transitions of a state that is stereotyped by «collection» either must have a trigger with a parameter that refers to an element of the collection (explicitly or implicitly by using the underscore), or has to be denoted by the stereotype «allItems».

*example for «collection» ( )*

Figure 4.13 depicts an example of a «collection». In the state `List` several actions are possible: on each `Item` (the `itemType` corresponds to a `Item` class from the content model) the `g()` or the `h()` can occur. For the identification of an item, the '_' notation is used in the parameter lists, instead of 'item:Item' (cf. extended version in figure 4.14).

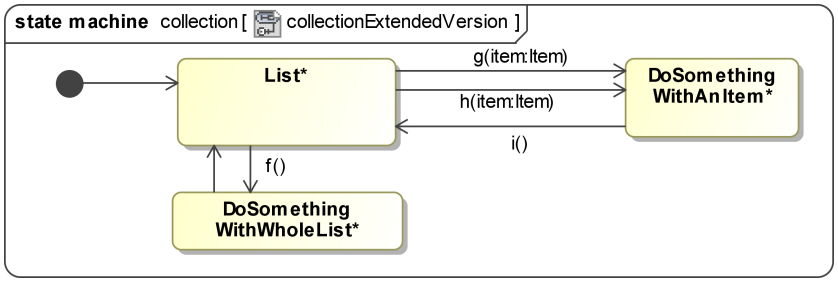


Figure 4.14: «collection» extended version

`f()` is an operation that is not executed for single elements of the collection, therefore the «allItems» stereotype is added to denote that `f()` is applied to the whole `List`.

#### 4.2.1.5 *Other Stereotypes*

This subsection introduces four additional stereotypes for typical situations in the web. Very common is the «search» ( ) and if a concrete semantics depends on the viewer for the web application (like a browser), the stereotype «browserDependent» is used. The stereotype «externalLink» is rather self-explanatory and used to express the navigation to foreign web applications. The specification of «integratedMenu» is more complex, but plays a part in keeping the arrangement of the diagrams clear, even if a complex navigation menu structure is used.

SPECIFICATION 11

*«search» ( )*

**«search» ( )** on transitions denotes that the output (usually a «collection») is determined by a search operation. The tag {expression} can be used to specify a kind of search expression in a formal or informal way.

*Constraint.* A trigger or effect such as `searchByName (name : String)` has to be added to specify the parameters of the search.

SPECIFICATION 12

*«browserDependent»*

**«browserDependent»** means that a behavior depends on the implementation of the browser.

*«externalLink»*

For example an «externalLink» on terminate states can on the one hand be opened in a new window or tab; in this case the pseudostate terminate is not activated. On the other hand it is possible to navigate to the external node by terminating the current web application.

*modeling navigation menus*

Before coming to the integrated menus, the usual way of modeling navigation menu should be outlined: to use a navigation menu in a web application means to substitute at least a part of a page with another content, which can include other navigation possibilities. This can be modeled using transitions from the border of a composite state to substates with the semantics of leaving the composite state, entering again and activating the target state. The content of the composite state is related to the part of the page that is exchanged and usually loses its context information when another menu item is activated.

SPECIFICATION 13

**«integratedMenu»** on transitions (from a composite state to inner submachine states) allows the modeler to split one menu into several submachine states without using entry points for each menu. This comes in handy if many roles and many menus are used. An example is depicted in figure 4.15.

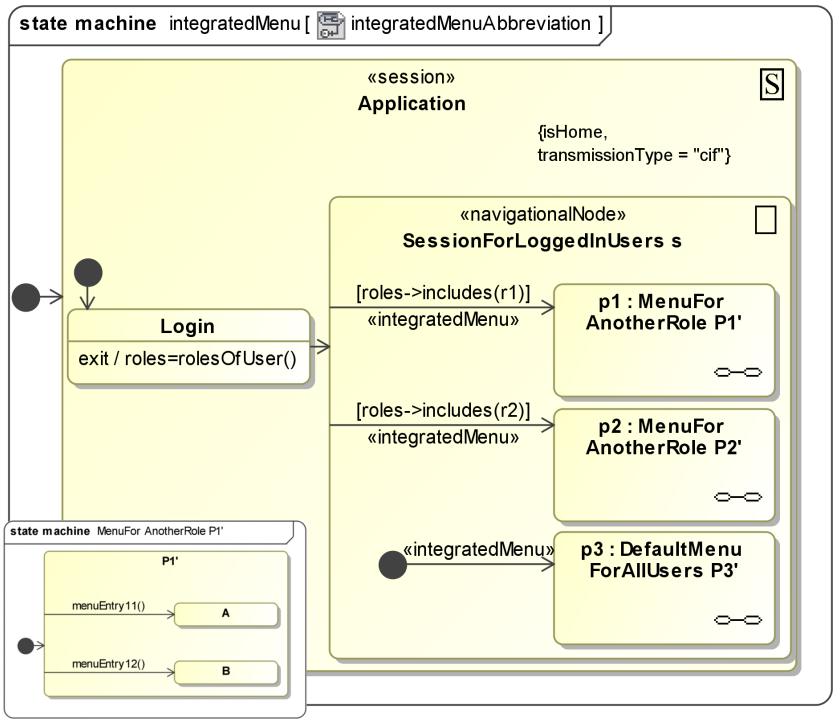


Figure 4.15: «integratedMenu» abbreviation

*Constraint.* A composite state s contains m submachine states $p_i \in P$ $(i = 1, \ldots, m)$. P is the set of submachine states, where transitions stereotyped by «integratedMenu» end. Transitions in the set T are typed by «integratedMenu» with the guard expressions $g_i$ and no other properties (like effects or triggers) and each $t_i \in T_i$ connects s or an initial node – see also the paragraph below – with $p_i$. Each $p_i$ refers to a state machine $p'_i$. Each $p'_i$ contains $o_i$ transitions $t'_{ij} \in T'_i (j = 1, \ldots, o_i)$ from a composite state $s'_i$ (which is the target of an initial state) in $p'_i$ to a substate $s''_{ij}$ in $s'_i$.

An initial state can exist in s. If no other transition is connected with a submachine state $p_i$ that includes menu transitions, which should be integrated, the transition coming from the initial node, pointing to $p_i$, has to be stereotyped by «integratedMenu».

*Definition.* For each $t'_i$, the transition $t_i$ that targets $p_i$ is copied, which results in the transformed transitions $\hat{t}_{ij}$ with the guards $g_{ij}$. Each $\hat{t}_{ij}$ targets a new entry point $e_{ij}$ of $p_i$ with the corresponding representation $e'_{ij}$ in $p'_i$. All $t'_{ij}$ are transformed to $\hat{t}'_{ij}$ that start at $e'_{ij}$ instead of $s'_i$.

For all transitions $\hat{t}_{ij}$, all properties of $\hat{t}'_{ij}$ are added. The guard $g_i$ is added with '&' to the guards $g_{ij}$ (for $j = 1, \ldots, o_i$). If one guard is empty, the '&' is left out. If a $t_i$ starts at an initial node, the «integratedMenu» stereotyped is removed, if not, $t_i$ is deleted itself.

All transitions without the «integratedMenu» stereotype remain unchanged.

*example for «integratedMenu»*

It is not easy to present a small and at the same time convincing example for an integrated menu, because the advantage of the use of this shortcut can be observed best in rather large and complex models. In figure 4.15 two roles are each allowed to access one substate machine. In addition, all users can use the default menu for all users, no matter what roles they take on. The first substate machine `p1` is depicted in the bottom left corner of figure 4.15 with two menu transitions targeting inner states. An inner initial node is not necessary, because of the «integratedMenu» stereotype at the transition that ends at the state `p1` in the context state machine `SessionForLoggedInUsers s`.

After transforming the diagram into a version without integrated menus (figure 4.16), entry points and the repetition of the guards for *all* navigation menus crowd the diagram. If `p1` had got 20 instead of 2 menus – e.g. for `r1`=administrators with the permission to access many configuration pages – 20 entry points with 20 copies of the guard `roles->includes(r1)` would be necessary at the top level. Consequently, changing all those guards would be a tedious task without the «integratedMenu» stereotype.

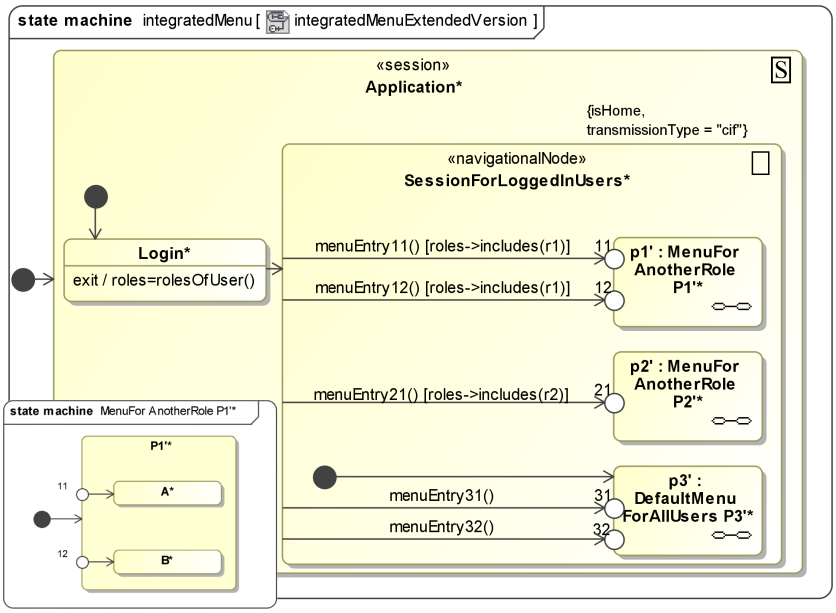


Figure 4.16: «integratedMenu» extended version

### 4.2.2 *Transformations between Navigation Class Model and Navigation State Model*

It is difficult to define a transformation between two models that do neither have exactly the same focus, nor the same expressiveness. Nonetheless, it is interesting to give a semi-automatic transformation between the UWE navigation class model and the *UWEsecurity* navigation state model a thought.

*"big picture" approach*

Knapp and Zhang created an "approach of building a UML state machine that integrates the separate concerns [...] of a web system into a big picture, which can then be validated formally". [26] In [26, chapter 3.1] they describe the transformation of UWE's navigation structures.

The difference to *UWEsecurity*'s navigation states model is that Knapp's state machine is simpler, as security features, user management, sessions and global menus are nonexistent. That's the reason why a *semi*-automatic transformation is presented in the following. Initially, the basic transformation is described and afterwards further options that can be chosen by the modeler are outlined.

*transformation from navigation classes to navigation states*

Basic transformation from navigation classes (see section 3.2.1.2) to navigation states:

NAVIGATIONCLASS ISHOME is transformed to a composite state with the same name and the stereotype «navigationalNode» with the tag {isHome=true}. An initial node is placed next to the new state and connected with it by a new transition. These elements are the starting position for all transformations.

NAVIGATIONCLASS ISLANDMARK equals to a navigation menu in the «navigationalNode» state with the tag {isHome}, if this is the outermost state.

INDEX correlates to a «collection» state. The «index» class itself is not transformed, but the item class that is connected with an association is converted to a «collection» state. The modeler has to specify further details after the transformation.

MENU is converted into 'menu' transitions that start at the border of a composite state and end at an inner state. Further information is needed to be able to identify the scope for each menu and to specify if two menus can be activated in parallel, e.g. if one menu item just expands something like a search field, but not changes the navigation possibilities of the rest of the page.

PROCESS CLASS. There are two options: it can be transformed into a state or into a transition with an action name equal to the process name. In some cases processes could also be effects, but a choice should be optional in order to provide a fast result at the first time. Default operation is to convert the class into a transition if there is exactly one outgoing «processLink» association and otherwise into a state.

QUERY is transformed to a «search» transition. If the «query» class is not connected with two directed associations, an incoming and an outgoing one, the modeler has to decide what should happen.

EXTERNAL NODE is converted to an «externalLink» terminal state.

NAVIGATION CLASS. Per default, a «navigationClass» is transformed to a new state, but the modeler should have the option to transform it to a transition or substate of another state instead.

Further refinements for the transformation could be:

USER MANAGEMENT. If user management should be introduced, the navigation classes have to be divided into groups: one group for each internal area with a guard that specifies the conditions for entering it. By default, this schema is transformed into an external visitor area (containing submachine states from the *UWEsecurity* patterns, e.g. "authentication via password form" for login) with transitions from the login state to an internal visitor area that internally uses the given guards and the «integratedMenu» stereotype.

NAVIGATION MENUS that cannot be derived from the navigation class model. Maybe a window is advantageous where the modeler can select elements from the traditional navigation class model. These elements can then be grouped together and be converted to a composite state with several menu transitions. This step should be optional, because presumably it is easier for the modeler if the transformation has less dialogs and instead the output includes only some composite states (see user management) and further states are located loosely within those states and are just connected with each other by "empty completion transitions" in the first place.

SECURE CONNECTIONS can be derived from the user management option to some extent: per default, the login and registration as well as all internal areas should be enclosed by a «session» state with {transmissionType='cif'}.

*transformation from navigation states to navigation classes*

The transformation is bidirectional, the other direction is described in the following. The navigation states model is more powerful, therefore information loss cannot be completely avoided. It is required that «target» stereotypes are replaced by applying their definition, before the transformation from the navigation states model to the navigation class model is executed.

NAVIGATIONAL STATES. The basis is the «isHome» tag, similar to the other direction of the transformation. All «navigationalNode» or «session» substates are unfolded, i.e. the inner states are converted to classes stereotyped by «navigationClass» (if no other rules are applicable, see below) and the composite state itself is transformed into a «menu» class (which belongs to a preceding class with a composition). In most cases, transitions can be converted to directed «navigationLink» associations.

COLLECTIONS are transformed into «index» classes with the name of the state concatenated with 'Index'. Those classes have a directed association (multiplicity * at the end) to a «navigationClass» with the original name.

EXTERNAL LINKS are converted to «externalNode» classes.

SEARCH transitions are equal to a new «query» class between the converted elements at the ends of the transitions. Directed associations typed by «processLink» are connecting the new class.

In summary, it can be stated that these two transformations are too complex to have a beneficial effect on the comprehensibility and speed of the process of creating another kind of navigation

model. Nevertheless, a limited implementation of the outlined transformations may be a good start.

## 4.3 UWEsecurity Patterns

Patterns are a common approach to tackle the problem of repetitive tasks. It is necessary to implement authentication due to its importance, as authorization is impossible without authentication. Otherwise everyone would have the same rights, because no distinction of users would be feasible. Therefore, several typical authentication related mechanisms are provided as patterns, e.g. user registration, authentication (login mechanisms), credential recovery (lost password), and further patterns, as user profile configuration.

### 4.3.1 *Registration*

*secure registration*

User registration has to ensure at least two things: On the one hand the user should be human, which can be verified by a CAPTCHA[3] as depicted in figure 4.17. On the other hand the information the user provides has to be useful, for example the given email address should be valid. Another frequent requirement is to encrypt the entered data during the transmission to the server, as determined by the tag {transmissionType='cif'}.

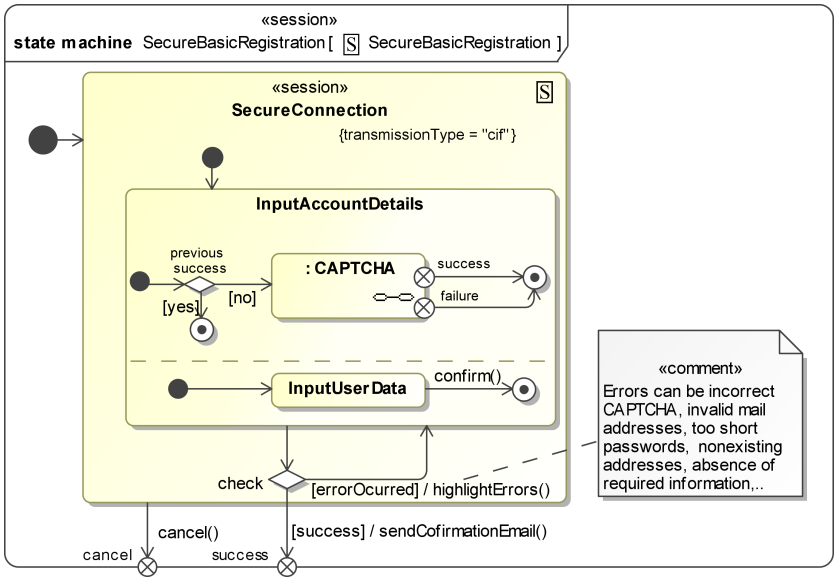


Figure 4.17: Secure registration

*insecure basic registration...*

Another variant is the insecure basic registration pattern. It does not specify a CAPTCHA or a secured connection and can later

3 An example is google reCAPTCHA. http://www.google.com/recaptcha, last visited 2010-12-20

be used e.g. for signing up with a nickname and an email for services that have not to be secure.[4] In addition, a typical registration process is available as activity diagram that distinguishes between

*. . . with additional activity diagram*

diagram clear, the semantics of the stereotypes «userAction» and «systemAction» has been enhanced so that both can also be applied to activity partitions. The advantage is that not every action in a stereotyped swimlane has to be stereotyped individually.

### 4.3.2 *Authentication*

After having signed up, the users usually want to log in. Several types of logon requests are provided as navigation state patterns:

- Login via password form, which is located at a website. Besides the navigation state machine, an activity diagram and an exemplary presentation are provided.
- Single Sign-on (SSO) with an additional sequence diagram for OpenID
- Login with a client certificate
- Digest access authentication, if man-in-the-middle attacks should *not* be excluded
- CAPTCHAs only, as for submitting comments to blogs

*authentication via password form*

Exemplarily the first two patterns are described in this subsection. Thus, figure 4.18 depicts the authentication via password form. Simple enough, a secure connection is established, the users can enter their names and passwords and after the submission the authentication can be successful or not. In this particular case error messages are displayed and the user can try it again. A cancel method is not provided, because this navigational pattern should be integrated as submachine state and in case navigation menus are used, the user can activate those outer transitions to navigate away from the login form.

*single sign-on with OpenID*

Another example of our predefined patterns that are presented here is the SSO as shown in figure 4.19. This sequence diagram[5] depicts the communication between client, site-server and OpenID provider. The main point is that the site-server only asks the OpenID provider to authenticate the user and afterwards relies on the authentication result.

Figure 4.20 depicts the corresponding navigation state diagram. Of course only navigation related states are shown, as the

4 An example is http://www.dict.cc/, last visited 2011-01-29, an online dictionary where e.g. a vocabulary trainer can be used after the registration.

5 For the creation of figure 4.19 information from http://www.theserverside.com/news/1364125/Using-OpenID, last visited 2010-12-15 was used.

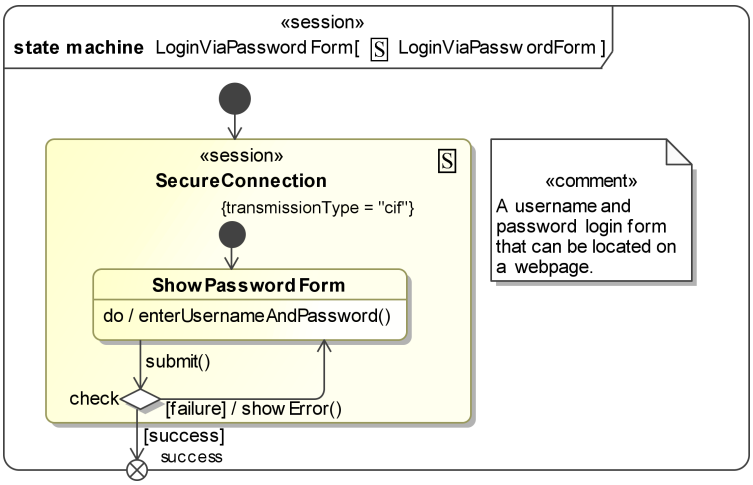


Figure 4.18: Authentication with a password form

possibility for the user to enter an OpenID URL and the vague representation of the authentication procedure itself. The result of the login is likely to change the further navigation flow of the modeled application.

### 4.3.3 *Credential Recovery*

*three patterns for credential recovery*

Credential recovery is important, if the user has lost e.g. his password. To recover the password, predefined questions can be asked (which is considered insecure, see section 2.1) or a SMS or email can be sent. The last one out of these three patterns is shown in figure 4.21. There is nothing special in this diagram except that the `SendMail` state is modeled even if it is only indirectly necessary for the navigation. 'Indirectly' means that the model would be incomprehensible without it and furthermore it might be desired to specify the recovery link in more detail.

### 4.3.4 *Further Patterns*

Our patterns should be self-explanatory so we only give some more examples that can be found at the attached CD.

*profile configuration pattern*

User profile configuration is something almost every web application offers that provides a login functionality. In other words users can choose from a typical navigation menu that is specified with a class diagram. An appropriate navigation states diagram depicts how to navigate through the tabs for editing account information, user details or privacy settings. The pattern also provides an activity diagram that focuses on the server actions. It is expectable that these diagrams can be used without further adaptation only on rare occasions, but they might be a good

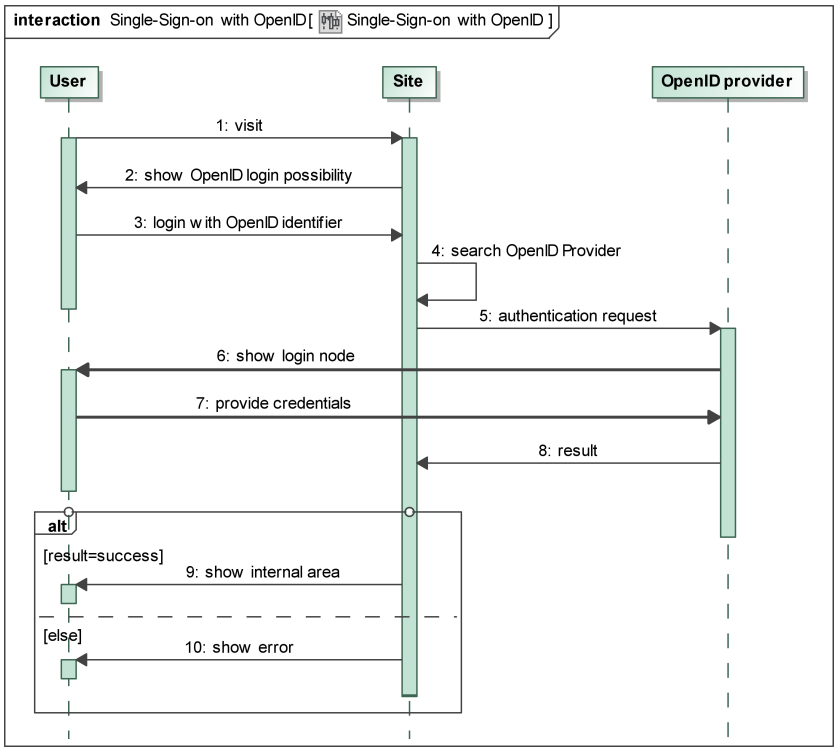


Figure 4.19: Authentication with Single Sign-on. Sequence diagram

start for the modelers, as they just can copy them into their own models.

*non-repudiation pattern*

Non-repudiation for web applications is treated as an orphan, because no common solution exists. Our "non-repudiation substitute" pattern points out that fact and suggests a combination of writing a log-file and sending a confirmation email to the user.

## 4.4 Role-Based Access Control

After having mentioned RBAC for session nodes in the previous sections, this segment introduces *UWEsecurity*'s role model, basic rights model and its transformation to SecureUML.

### 4.4.1 *Role Model*

A role model is a special case of user model, in which characteristics of the roles users may take on are specified. *UWEsecurity*'s role model comprises instances from a `Role` class or enumeration and models their hierarchy and connection to `User` instances with links. Usually the moment after the installation of the web application is captured in this way to avoid inaccuracies because of subsequently registered users. Detailed examples can be found in section 6.2.2 (roles as items of an enumeration) and section A.3 (roles as classes). As a matter of course more than one snapshot

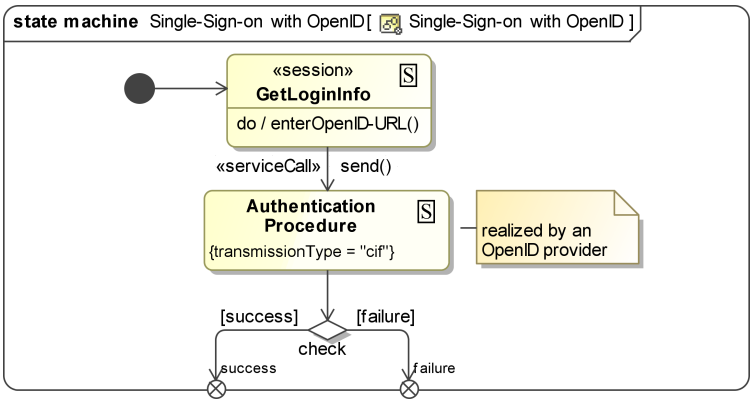


Figure 4.20: Authentication with Single Sign-on. State machine diagram

can be modeled e.g. in case the application allows to change the role structure dynamically.

In a model-driven process, we may generate a first version of the role model from the actors hierarchy in the use case diagrams. The only thing we need to know is which classes in the user model correspond to the typical RBAC `Role` & `User` classes including the association that represents the `inheritsRightsFrom` relationship, if a hierarchy is used.

### 4.4.2 *Basic Rights Model*

*Role Model and Basic Rights Model instead of SecureUML*

At the beginning we planned to use SecureUML together with the dialect ComponentUML (see section 3.1.2) for expressing RBAC, but SecureUML introduces a new class for each user, instead of instantiating one `Role` class or enumeration as generally used in implementations. Furthermore, association classes are not popular, due to the fact that they are unhandy and not supported by many CASE tools, and the repetition of e.g. method names in order to define the access is cumbersome. In addition, it is somewhat peculiar to create new classes for the return values within the association classes, each time the model is used, or select them without ordinary tool support from a UML profile.

*basic rights model*

By contrast, in the basic rights model the role instances are connected with the content (or user model) classes or their attributes or methods just with stereotyped dependencies. The stereotypes to specify RBAC with dependencies are defined in figure 4.22.

*RBAC for classes and their attributes and methods*

The basic rights model is based on CRUD (create, read, update and delete) with an additional execute operation. This means the users are only allowed to execute a method of a content class, if they are associated (at runtime) with a role that is connected to this method with an «execute» dependency or with this class with an «executeAll» dependency (if this RBAC is modeled at all, see 60). The stereotypes «read» and «update» are used in a similar way, but for attributes instead of methods. The «delete» and «cre-

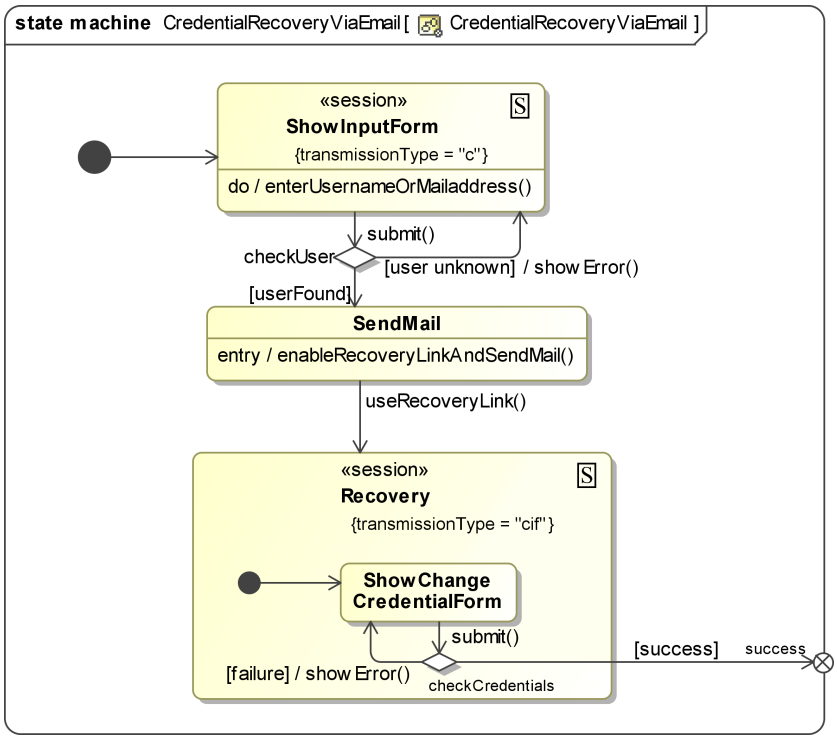


Figure 4.21: Credential recovery via email

ate» stereotypes refer to the whole object and their specification is especially useful if the object should relate to a database entry of the web application. Additionally, authorization constraints – that are similar to those of SecureUML – may be added as comments. For the sake of clarity, they can be connected with corresponding dependencies.

Two important issues came up when conceptualizing the basic rights model:

*the «~All» stereotypes*

First, the impact of the introduction of {except} tags for the «~All» stereotypes have to be considered. On the one hand, a model that permits and denies access at the same time can become confusing, as for large applications several diagrams may be used and it is difficult to see inconsistencies. On the other hand, this extremely rare case could be checked with a tool and there is no good reason for modeling nineteen «read» dependencies if the modeler wants to express that the twentieth and latest attribute cannot be accessed.

*when to assume default permissions*

Second, the claim for default permissions when nothing is specified. Our first thought was to allow nothing in this case, because no careless mistakes can occur when a new attribute or method is added in the content model without adapting the basic rights model. In spite of that, practice has shown that it is not useful to specify the exact permissions for *all* classes. It is far more important to have the freedom to specify RBAC only for well-chosen classes. As a result, we defined that if a content

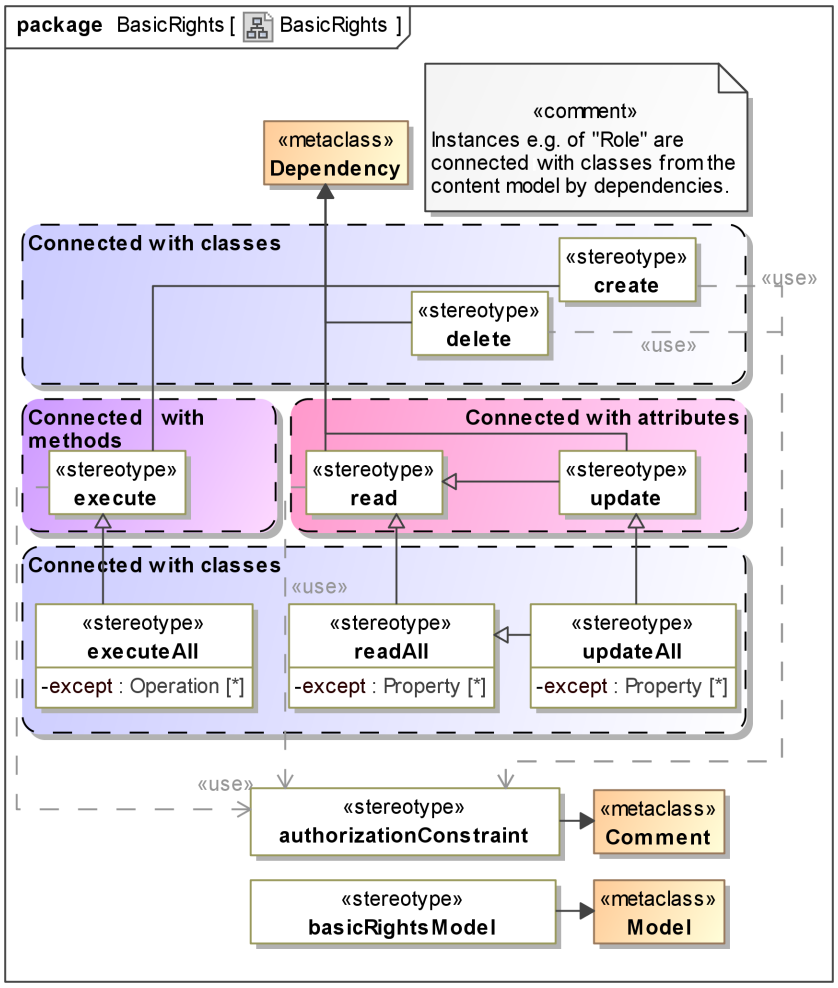


Figure 4.22: BasicRights UML profile

class occurs in the basic rights model and for some *methods* of a class RBAC is modeled and for others not, the permission should be denied for the undefined ones. Same with attributes and the operations create / delete.

Again, the trick is to demand a full specification for a single class (that is shown in the basic rights model) in the categories:

- create/delete
- execute/executeAll
- read/readAll and update/updateAll (as usual, update includes read)

This means the modeler is free to qualify only some classes and if e.g. only the execution of methods is important for him, nothing can be derived for the access on attributes. Concrete modeling examples are located in sections 6.2.3 and A.4 and the following subsection (section 4.4.3) outlines the transformation to SecureUML.

### 4.4.3 *Transformation to SecureUML with dialect ComponentUML*

The disadvantage of the invention of an own model – as our role model and basic rights model – normally is that existing tools for more popular modeling techniques cannot be used. A transformation to SecureUML (with the dialect ComponentUML) solves this problem for *UWEsecurity*. SecureUML's metamodel is described in section 3.1.2 and as can be derived from the last two sections, our models are identical with regard to the expressiveness, since another representation cannot change the meaning of the model. The following list sketches the transformations in both directions between *UWEsecurity*'s role and basic rights model and SecureUML.

ROLE AND USER ELEMENTS AND THEIR HIERARCHIES Role and user instances are converted to «Role» and «User» classes. The 'inheritsRightsFrom' link is replaced with a generalization in SecureUML and vice versa.

AUTHORIZATION CONSTRAINTS The stereotype «authorizationConstraint» has a capital letter in SecureUML.

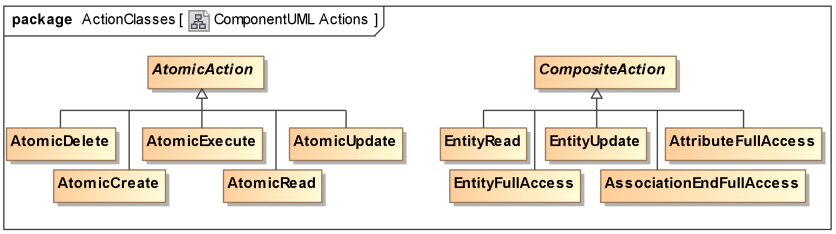


Figure 4.23: SecureUML. ComponentUML actions

DEPENDENCIES The dependencies are replaced with «Permission» association classes (between the content and the «Role» elements), which contain adequate "access specification methods". Possible 'return values' for RBAC are depicted in figure 4.23 («~All» equals '~FullAccess'). In the other direction, some constraints can be simplified with «~All» stereotypes, by using {except} tags, if there are more single items to grant access to than to deny. In ComponentUML a distinction is made between association ends (which may relate to roles) and attributes. In the basic rights model, roles and attributes are treated equally.

CONSTRAINED CLASSES The classes are the same, but the meaning of omitted information could differ.

# 5 IMPLEMENTING UWESECURITY MODELS

After having modeled the web application with *UWEsecurity*, there are several ways of implementing the application. This chapter describes the realization of security features like authentication and RBAC on content or user model classes and on navigational nodes and addresses the implementation of secure connections.

The focus is on the relationships between these security aspects, but beforehand the next table gives an overview over the modeling elements and their implementation. *realization overview*

| **Security Feature** | ***UWEsecurity* Model** | **General Implementation** |
|---|---|---|
| Authentication | pattern(s) in the navigation state model | web framework patterns |
| RBAC on classes | BasicRights model | individual implementation related to database access control |
| RBAC on navigational nodes | navigation state model with «session» states and tags {rolesExpression}, {roles} | rules and specification of a presentation on error in the web framework; `.htaccess` files |
| Secure Communication | navigation state model with «session» and secure {transmissionType} | establish appropriate TLS \ SSL connection (see web server configuration) |

## 5.1 Authentication

For several authentication related patterns of *UWEsecurity*, a counterpart exists in many web frameworks, as a predefined registration, login and lost password form. Sometimes the developer can also use modules from the web framework community for additional features like SSO and authentication with client certificates. This is especially useful if a standard implementation is not available or insufficient. *implementation of authentication related patterns*

Authentication goes hand in hand with user management and RBAC, therefore those parts have to be co-ordinated. *relationships*

## 5.2 Role-based Access on Classes

RBAC on content or user model classes, as it is defined in the *UWEsecurity*'s basic rights model, often is not directly supported by web frameworks. That is the reason why the models should be used as basis for the implementation. In addition, the access control structure can be distributed thus many classes contain methods that decide whether object or database entries are accessible or not.

## 5.3 Role-based Access on Navigational Nodes

*correlation of RBAC on classes and nodes*

The RBAC on classes is often directly connected with RBAC on navigational nodes, since a user should not be able to access navigational nodes that link to a page where protected information is expected to be shown. If the user navigates to such a page directly, usually an error message is displayed, but it is user-friendly if this error state can only be reached by accessing a URL and not while navigating through the application. A similar scenario is the access restrictions for method execution: it is unpleasant to receive an error message after the invocation of an unauthorized method and it is more professional to hide that option right from the start.

*realization*

To implement the «session» states with the tags {rolesExpression} and {roles}, primarily the latter tag has to be integrated in the {rolesExpression} (see specification 7 in chapter 4). Afterwards, these roles have to be used for the specification of page access in a web framework and finally an alternative presentation (e.g. an error message and/or a redirection to the login form) has to be set.

Another possibility for the customization of navigational nodes are `.htaccess` files. They can not only be used to rewrite URLs to shorter ones, but also to block users by domain or by IP address. Furthermore security restrictions for the folder that contains the `.htaccess` file can be specified.

## 5.4 Secure Communication

*link to other security aspects*

All previous subchapters try to protect some kind of information. But if the access can be denied for users that are regularly using the application, it does not automatically mean that only authorized users can read the critical data. That is because web applications are client/server applications that transmit data between the server and the client (for static web sites only once, but the trend is toward RIAs that can communicate almost continuously with the server). This data transmission may be eavesdropped by

a third party and thus many of the other security precautions are bypassed.

*realization of secure connections*

As already mentioned in chapter 2, the common solution for «session» with {transmissionType='cif'} is the use of X.509 server certificates and a secure connection over TLS / SSL and https. The concrete setup can be adjusted in the configuration file of the web server. Another approach is to encrypt / decrypt sensitive data in the client and server part of the application manually, but non-standard operations are not recommendable, as the new software can hardly be as bullet proof as established standards.

The following part of this work gives illustrative examples of working with *UWEsecurity* in practice. Among other things the implementation of a case study is described in chapter 7.

# Part III

# WORKING WITH UWESECURITY

# 6 CASE STUDY – DESIGN OF HOSPINFO

When dealing with medical data, several aspects of anonymity, pseudonymity laws and provisions have to be taken into consideration. Authentication, authorization and secure connections are basic and important requirements for Hospital Information Systems (HISs). The sooner security aspects are incorporated into the development process, the sooner errors and inappropriate concepts can be revealed that otherwise would have required costly fault analysis and patching cycles.

Our case study, called *HospInfo* (Hospital Information), is a web-based HIS. In this chapter, we elicit and specify the requirements and model the application using *UWEsecurity*, introduced in the previous part of this thesis.

## 6.1 Requirements Analysis

In this section, the requirements of *HospInfo* are discussed, especially why to use a web application, the general structure of *HospInfo* and its security features.

### 6.1.1 *Examples of Hospital Information Systems*

Web applications have the advantage of being universally accessible, because the users only need a browser and eventually some plugins. In the following some HISs are introduced that are used in the area of Munich.

*web-based example: LAMP-IS*

In the hospital "Klinikum der Universität München"[1], *LAMP-IS* [2] (information system) is used, which is a web-based HIS. LAMP-IS originally stands for the applied techniques: Linux, Apache, MySQL and PHP. It has been designed and programmed largely by Dr. Endres since 1999 for the Intranet of the hospital. Due to the heterogeneity of systems in the medical environment, it is difficult to install a local software on the clients. Usually even diverse browsers are used.

The problem of heterogeneity can be tackled by using web applications for the client and the heterogeneity of (server) applications is tried to be overcome by standardized protocols e.g.

1 Klinikum der Universität München. http://www.klinikum.uni-muenchen.de, last visited 2010-10-24

2 LAMP-IS KIS (Krankenhausinformationssystem). http://www.klinikum.uni-muenchen.de/Medizinische-Klinik-Innenstadt/download/de/HeiseOpen08_LAMP-IS.pdf, last visited 2010-10-23

Health Level 7 (HL7) and Digital Imaging and Communications in Medicine (DICOM)[3]. The physicians can watch a preview of nuclear magnetic resonance imaging (NMRI) images in LAMP and have access to e.g. full 3D scans, stored in a separate Siemens[4] picture archiving and communication system (PACS) via a web-client (Siemens syngo Imaging). That client does not require an installation as an administrator and can use the authentication provided by LAMP. In general, there is the possibility to use browser plugins, but apart from the lack of availability of those plugins for medical purposes, it is cumbersome or even impossible to install desktop applications (e.g. a special browser in a particular version) on all PCs, especially when access to the HIS over the Internet is supported.

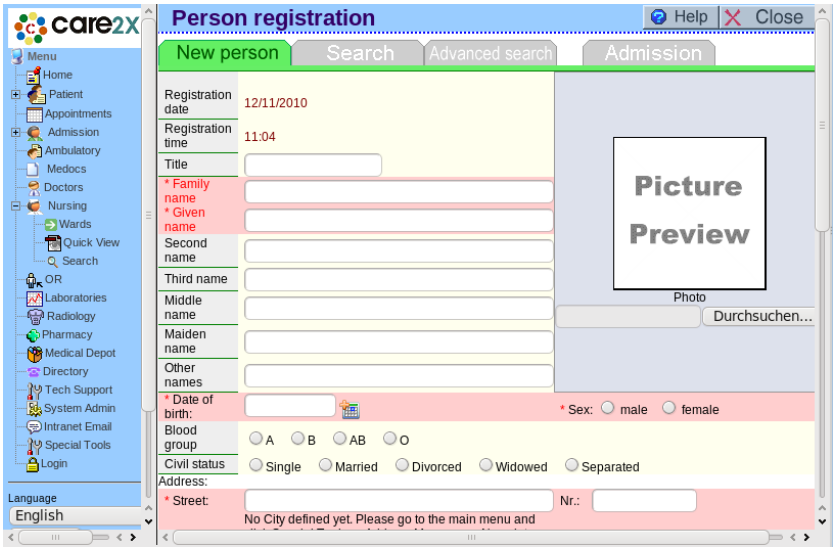


Figure 6.1: Care2x. Person registration.

*web-based example: care2x*

Another web based HIS is *care2x*.[5] It is an open source HIS, also written in PHP. There are several translations available and among other things the code was updated from PHP4 to PHP5 in the last year. An installation manual[6] is available for Ubuntu, Nginx WebServer, MySQL and PHP, but we experienced no serious problems using Apache instead of Nginx while testing the HIS. Care2x implements a lot of functionalities therefore we decided to develop *HospInfo* with basic functionalities in order to keep our case study precise and coherent. An example is the

3 HL7 and DICOM are standards for handling and transmitting medical data. Further information can be found in [22, chapter 6.4] and in [29, chapter 10 and 13]. [22] is a good and relatively new book to learn about the interaction of IT and management and healthcare.

4 Siemens Healthcare. http://www.medical.siemens.com/, last visited 2010-10-20

5 care2x. http://www.care2x.org/, last visited 2010-08-26

6 Installing care2x 2.6. http://sourceforge.net/apps/phpbb/care2002/viewtopic.php?f=6&t=6, last visited 2010-09-15

"person registration" form of care2x: it includes many text fields – as shown in figure 6.1 – but the amount of input is not related with the amount of security features we want to model, so we can keep it short for our purposes.

*non web-based example*

Many HISs are not web-based (yet). An example is the *NEXUS / HIS*[7], which is used in the hospital "Marianne-Strauß-Klinik".[8] It requires to execute a desktop application and the available modules have to be paid adequately to requirements, usually per year. Of course the disadvantage of this system is the limited flexibility.

*conclusion with regard to web-based HISs*

In summary it can be said that web applications are easy to use and require less maintenance effort than desktop-based solutions. Nevertheless, some viewers usually have to be installed locally and the most important problem (similar to desktop-based versions) is the integration of already existing software components, e.g. radiology information systems (RISs). Furthermore, before using the software, local regulations and requirements have to be validated. For our case study, we abstract from the aspect of software heterogeneity in hospitals and concentrate on the web application itself.

### 6.1.2 *Functionality of HospInfo*

For our case study we consider administrators, physicians, nurses and receptionists as the most important user groups, which are represented as actors in the use case diagram shown in figure 6.2. The following functionality is required for *HospInfo*: Staff members should be able to register by providing a forename and a surname, a well-formed email address and a password, which has at least the length of five characters and which has to be entered twice to avoid spelling mistakes. The roles of the staff members are set later by an administrator, who also chooses the ward (or main workspace).

*staff*

Consequently, *registered visitors* initially have no special permissions, except editing their profile, e.g. changing their name in case of marriage or spelling mistakes and their email address. They can also set a new password (which requires entering the previous one first). Furthermore, a logout option is required. If someone forgets his password he should enter his email address to receive a new one. Logged in or not, the home page and a page with general information, such as useful links and phone numbers should be accessible for everyone, as depicted in figure 6.2, the UML use case diagram of *HospInfo*.

*patient records*

*Physicians* need the permission to create new patient records or change information of patients. The patient information can

7 NEXUS AG. http://www.micom-medicare.de, last visited 2010-10-24
8 Marianne-Strauß-Klinik. http://www.ms-klinik.de/, last visited 2010-10-24

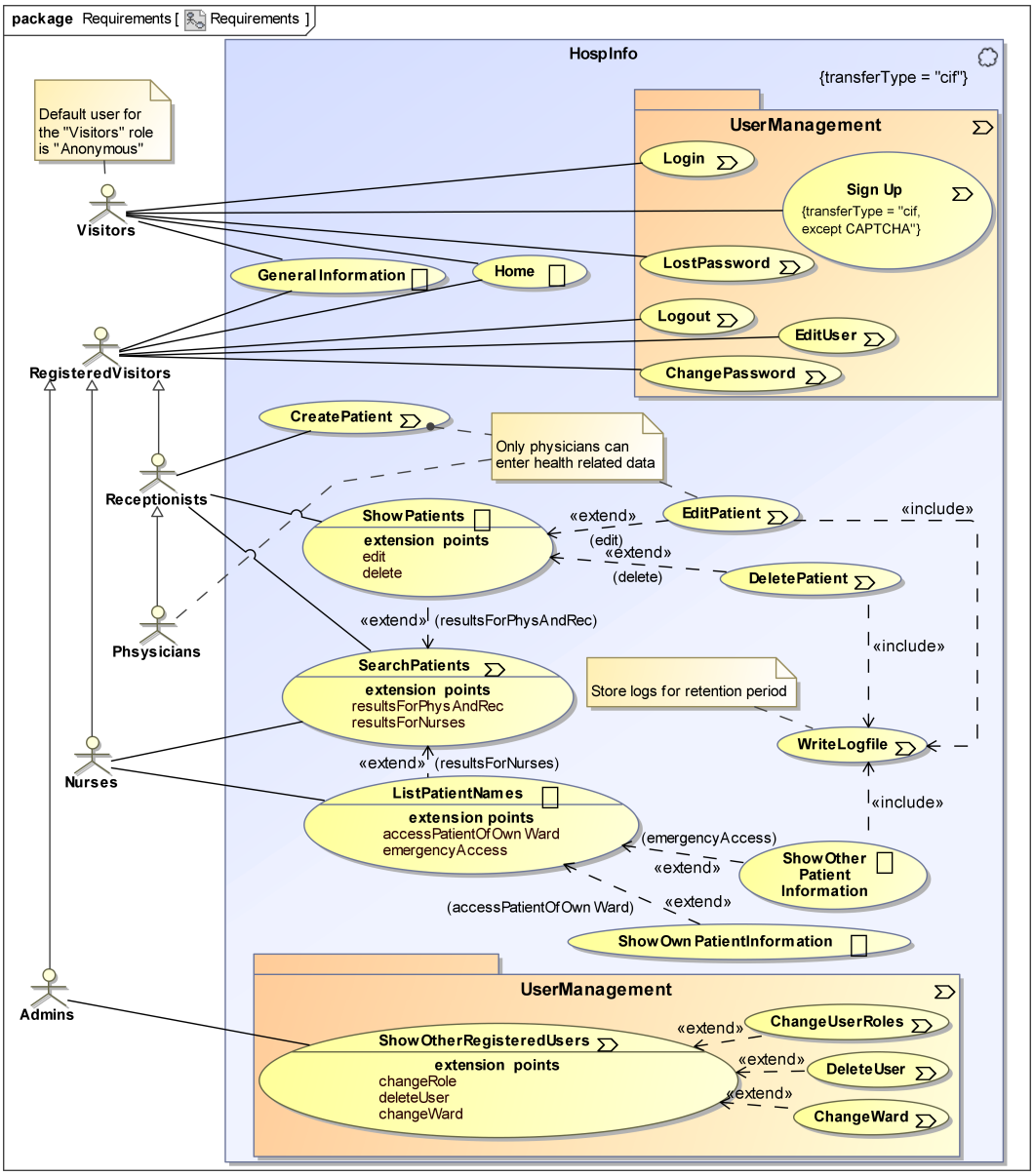


Figure 6.2: HospInfo. Use case diagram

be read in a list of all patients. This would not be convenient for normal sized hospital, therefore a search function is necessary. Each patient record should comprise name, gender, date of birth, address, blood group (A, B, AB or 0), the affiliation to a ward and medical information, given by diagnosis-related group (DRG) codes or free text.

*Nursing staff* should be able to read the information about patients from their ward and access the health records of the patients from other wards in emergency (which differs in the fact that the access is logged). For this purpose, two lists of patients should be presented: one for the patients of the same ward as the nurse and one for other patients (cf. use cases `ShowOwnPatientInformation` and `ShowOtherPatientInformation` in figure 6.2).

It should not be the task of physicians to enter organizational patient data, usually this is done by *receptionists* who can read all information (or at least accounting relevant parts), but cannot change health related data, because they normally have no medical education.

In figure 6.2 it can be seen that use cases like `Edit-` and `DeletePatient` are stereotyped by «processing» (⌦), whereas use cases that do not change the state of the application are annotated with «browsing» (□). Additionally, stereotypes can be omitted (as described in further detail in section 4.1), if they are assigned to a parental package, which in this case is used for the `UserManagement` package.

*prototype vs. ready-to-use application*

*HospInfo* should become a prototype to demonstrate the modeling and implementation of security features (see next subsection). An application for a real world scenario would include more features, the patient registration would be separated from the admission and the former visits would be saved together with the medical report as well as the treatment, e.g. medication or surgery and the linked resources, e.g. blood values or diagnostic images. Usually there is a calendar and there are dedicated views for each section, e.g. for radiology, pharmacy and for laboratories. Every staff member also has an own calendar and some entries have to be synchronized, e.g. appointments should allocate time in the calendar of a patient as well as in the one of the attending physician.

### 6.1.3 *Security Features*

We have chosen our case study with regard to the manifold security aspects that are important for a HIS:

- A **secure user management** ensures that no credential theft occurs. In *HospInfo*, the password should be entered into an HTML password field (the characters are not shown) and the credentials should always be transmitted in a secure way.

- Therefore, **secure transmission** is required not only for sending the credentials, but also for retrieving medical data. Thus, freshness, confidentiality and integrity are crucial for a HIS, as it is easy to be liable to prosecution, when confidential data gets into the wrong hands. Consequently, figure 6.2 shows the tag {transmissionType='cif'} for the «webUseCase» package to specify this requirement.

- The login session should provide a **session timeout** so that the users have to re-authenticate themselves after a certain amount of time. This is necessary, because medical staff is

often called off in an emergency without having the time to log out.

- **Access control** should be available, as described in the previous subsection (6.1.2). RBAC supports the following roles that correspond to actors in the use case diagram: admins (full access), physicians (read and write access to all patient data), receptionists (can read all patient information, but are just allowed to edit non health related entries), nurses (read access regarding patient records). As depicted in figure 6.2, the *break glass policy* is used to allow logged access for nurses in case of emergency, e.g. if a nurse is called to help a patient from another ward and it is required to access the patient record.

  Access control does not only refer to the access the users have on objects, but also to the *web pages* they are allowed to see in their role. This is tackled in two ways: on the one hand, unavailable options should not be shown; on the other hand, an *access denied* error message should indicate that the user has to login with the appropriate permissions to display the requested page. This happens especially, if the user tries to access parts of the application directly, using a URL. An example would be a nurse trying to access the patient creation form, using the URL `https://host/createPatient`.

In this way *HospInfo* demonstrates typical secure workflows: user registration, authentication and authorization, password recovery (or change) and session or error handling are addressed.

## 6.2 Modeling

This section illustrates the use of *UWEsecurity* by presenting the software design model of the case study *HospInfo*. We have already shown a use case diagram in the last section, so we can now concentrate on the other UWE and *UWEsecurity* views (content, user and role model, basic rights, navigation states, traditional navigation classes, presentation and process). For a more detailed overview of the views and their intention, the reader is referred back to figure 4.1 on page 35.

### 6.2.1 *Content Model*

When the modelers know the main concept of the future application, they usually start with the content model to capture the involved elements as UML classes. In *HospInfo* the focus of interest is on the `Patients` with some attributes as `name`, `address`, `ward`

or `gender` (see figure 6.3). If the item `administrative` is chosen for the patient's ward from the `Ward` enumeration, it means that a patient has been created for the purpose of software testing.

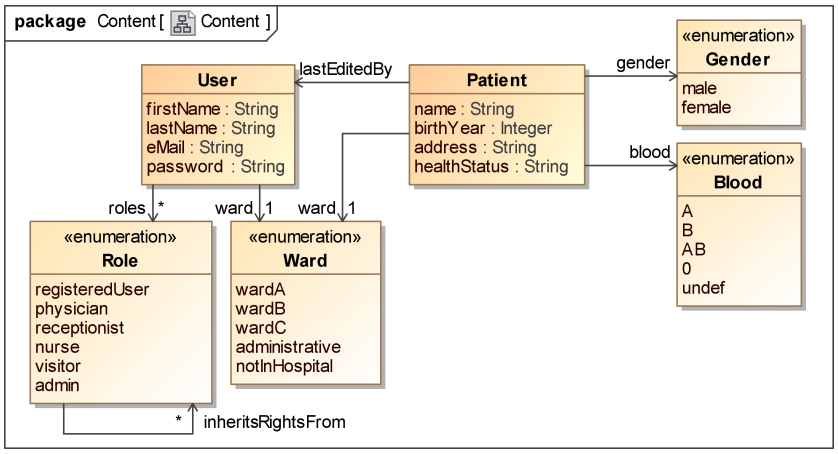


Figure 6.3: HospInfo. Content diagram

*connection to the user model*

The associations to `User` and `Role` are shown in the content model even if both classes are located in the user model as described in the next section.

### 6.2.2 *User Model and Role Model*

In the user model the class `User` is accompanied by the enumeration `Role`. We have chosen an enumeration with regard to the future implementation with the web framework Lift.

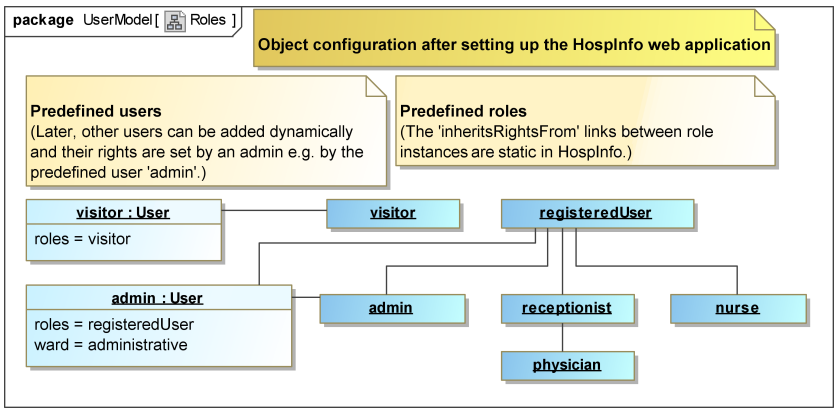


Figure 6.4: HospInfo. Role model

The user model includes the almost self-explanatory role model (cf. section 4.4.1) that is depicted in figure 6.4. The UML association role `inheritsRightsFrom` is not shown in the enumeration items, because MagicDraw does only support the link presentation between those items.

### 6.2.3 *Basic Rights Model*

Figure 6.5 depicts the basic rights model of *HospInfo* with access specifications for the classes `User` and `Patient`.

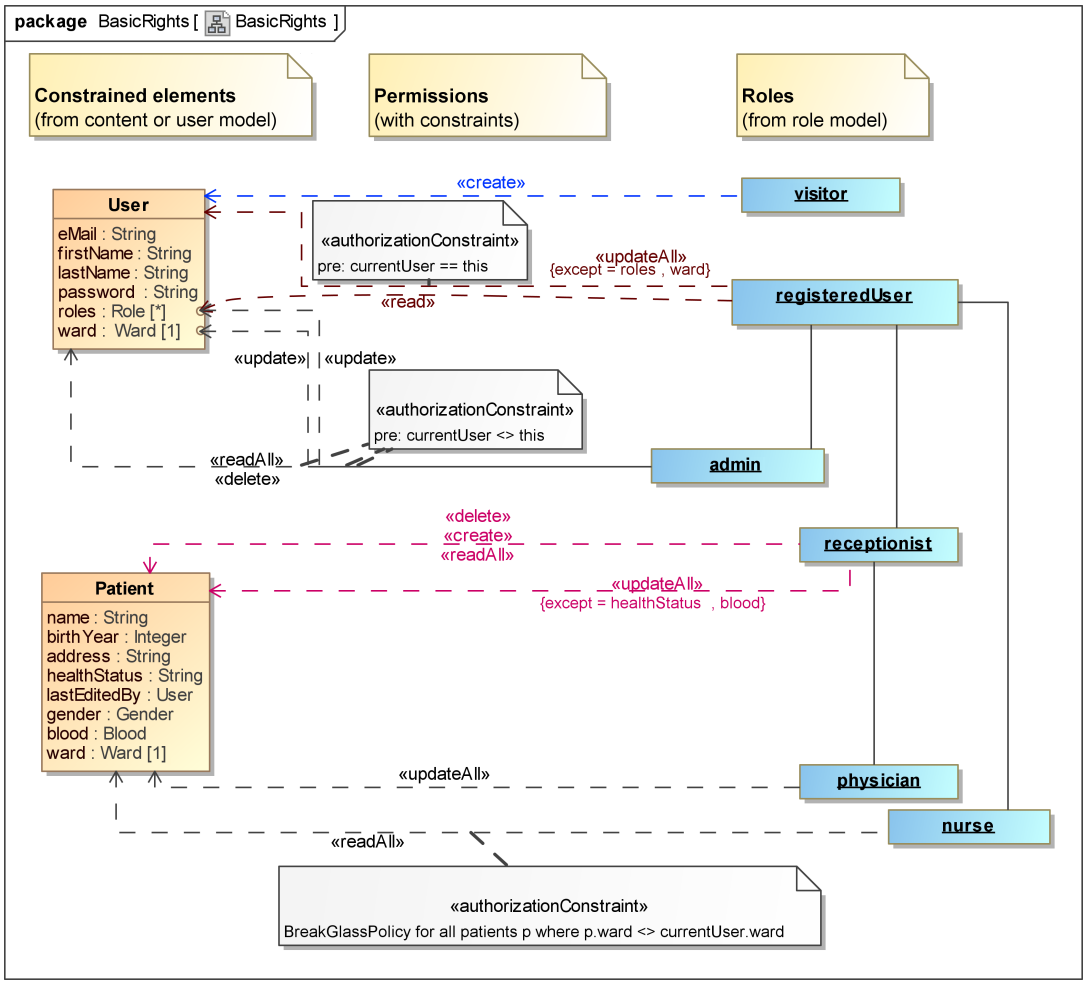


Figure 6.5: HospInfo. Basic rights diagram

The rule that `admins` cannot change their own user account is depicted with the «authorizationConstraint» in the center of the diagram (cf. section 4.4.2). Thereby the 'variable' `currentUser` stands for the operating administrator.

The {except=healthStatus, blood} tag on the «updateAll» dependency between the `receptionist` and the `Patient` specifies that the updates on all other attributes of the class `Patient` are permitted. By contrast physicians can «updateAll» `Patient` attributes without any {except} restrictions.

For nurses, the Break-Glass Policy (BGP) for patients of other wards is captured in a rather informal «authorizationConstraint» (see bottom of figure 6.5).

### 6.2.4 *Navigation States Model*

*navigation menu*

The *UWEsecurity* navigation states model consists of a set of menus (see figure 6.6) that are connected with the state machines. Abstract menu classes as `DefaultMenu` ( ) cannot stand for themselves. That means there is no menu that only presents those five menu entries. *HospInfo* offers eight menu entries to an external visitor: the five ones from the `DefaultMenu` and three additional ones that belong to the `Visitor` class.

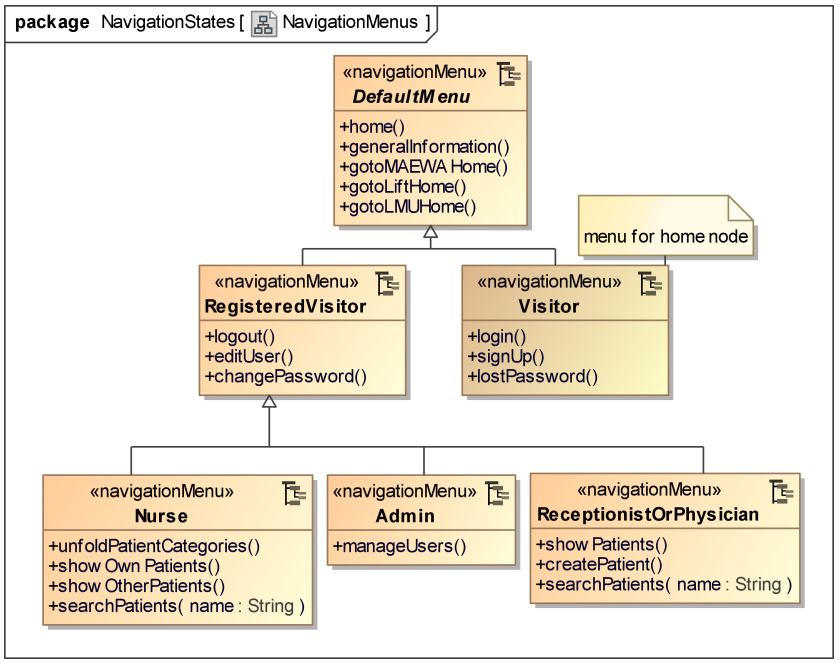


Figure 6.6: HospInfo. Navigation menu

*navigational state machines*

The names of the menu classes give a clue where they are finally used, but the concrete connection is set up in the navigational «session» ( ) states (cf. section 4.2.1). Figure 6.7 shows the main navigation state diagram for *HospInfo*. At the bottom of that diagram some external links can be found ( ). They correspond to the links that are always displayed in the footer of the application. The whole application `HospInfo` transmits all information in a confidential way and cares for the integrity and the freshness of the data (denoted by the tag «session» {transmissionType='cif'}).

Basically *HospInfo* consists of two navigation areas that are depicted in figure 6.7: a visitor area (in the diagram on the left) and an internal area (on the right), which is guarded according to the existing roles. The guards on the transitions are needed to specify the available menus and the «session» {roles} tag allows the modeler to define a set of roles that each can access the state.

In the visitor area the *UWEsecurity* patterns are used as substate machines e.g. for the secure login via password form and for the password recovery. The triggers of the transitions from the

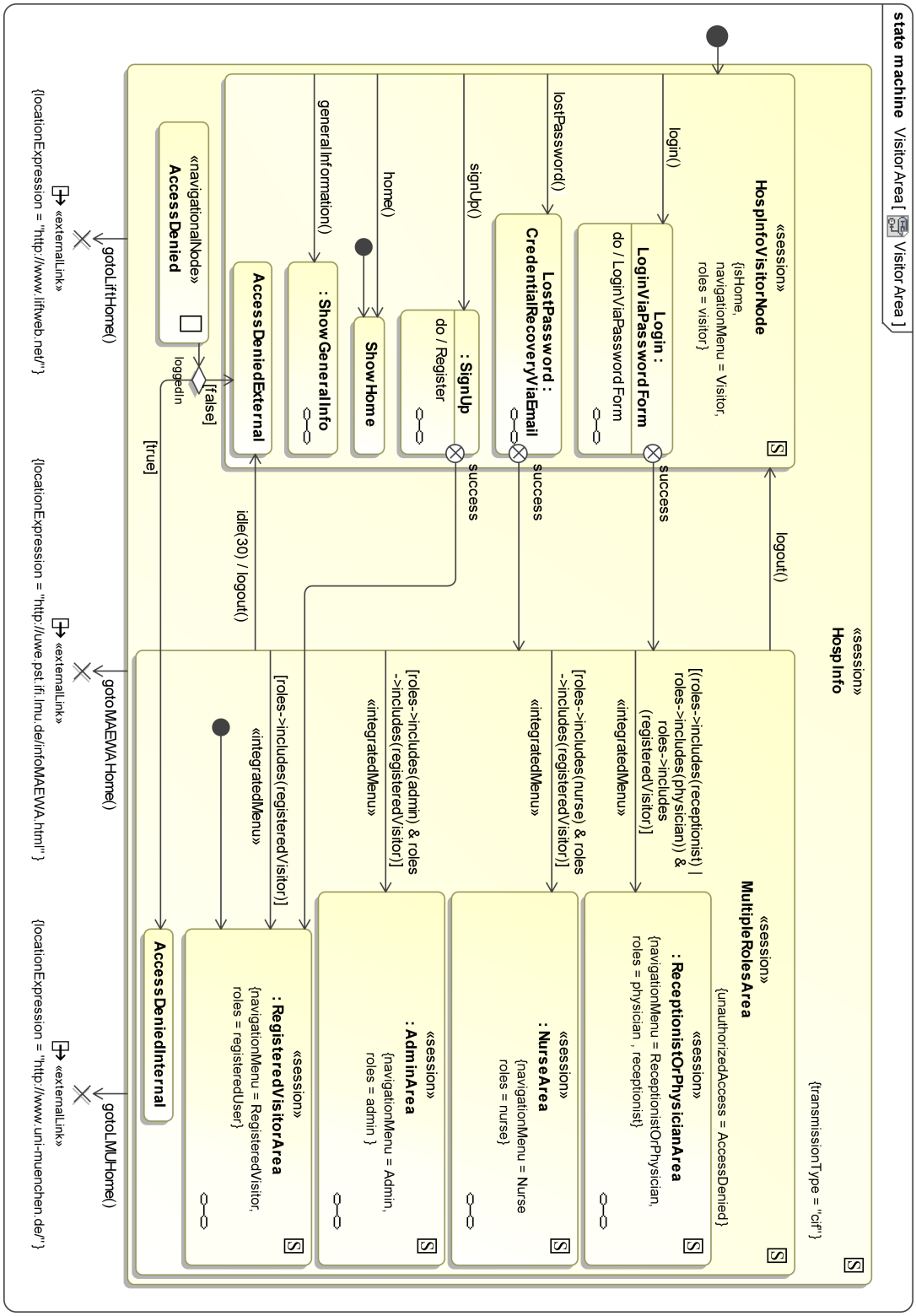


Figure 6.7: HospInfo. Navigation diagram of visitors

visitor session to substates are linked with operations from the «navigationMenu» classes.

After the successful login, the menus that should be available for the user are constructed with guards and the «integratedMenu» stereotype. In case of unauthorized access the `AccessDenied` state is activated. If the user is logged in, an internal error node becomes active, if not an external message is displayed, which differs from the navigational perspective at least in the navigation menu that is available.

An example for an actual internal node for receptionists or physicians is depicted in figure 6.8. Because of the integrated menus, the three triggers `createPatient()`, `showPatients()` and `searchPatients(...)` are removed from the inner transitions and combined with the first transition in the `MultipleRolesArea` state, as described in specification 13 (see figure 6.7). After applying the transformation according to the specification, three transitions connect the `MultipleRolesArea` state with three new entry points of the topmost substate machine.

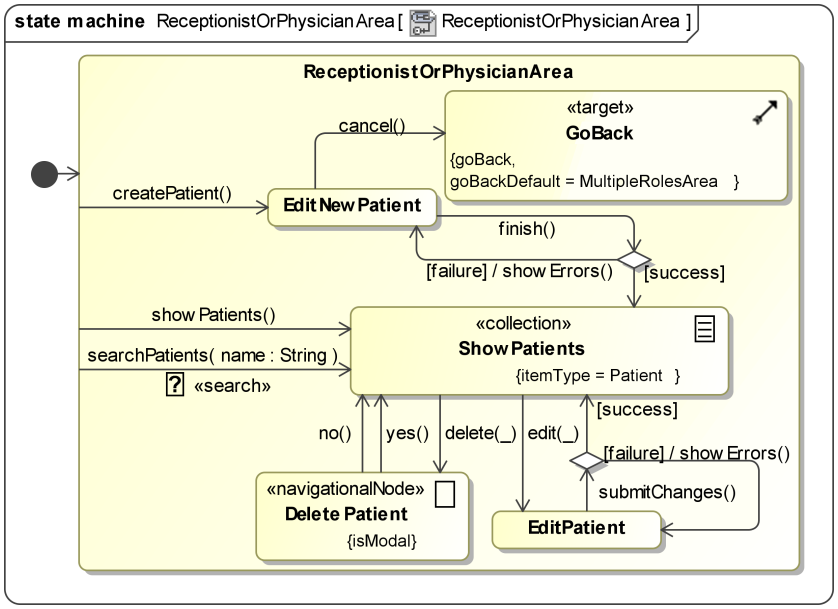


Figure 6.8: HospInfo. Navigation diagram of receptionist / physician area

Another feature of the diagram in figure 6.8 is the use of the «target» (↗) stereotype. It makes sure that after canceling the creation of a new patient *HospInfo* navigates back to the previous page of the application (or to the `MultipleRolesArea`, if the patient creation had been accessed directly).

Furthermore, the `ShowPatient` state is a «collection» (🗏) of `Patients` and due to no outgoing transition is stereotyped by «allItems» all trigger methods refer to a patient element, e.g. the `edit(_)` method stands for `edit(p:Patient)`.

### 6.2.5 *Navigation Classes Model*

To be able to draw a comparison between the new navigation state diagram and the traditional navigation class diagram for *HospInfo*, the latter is depicted in figure 6.9. The modeling of authentication issues is not provided, but menus (▤) as the `ExternalMenu`, `AdminMenu` etc. allow to draw conclusions to the substate session areas that are depicted in figure 6.7.

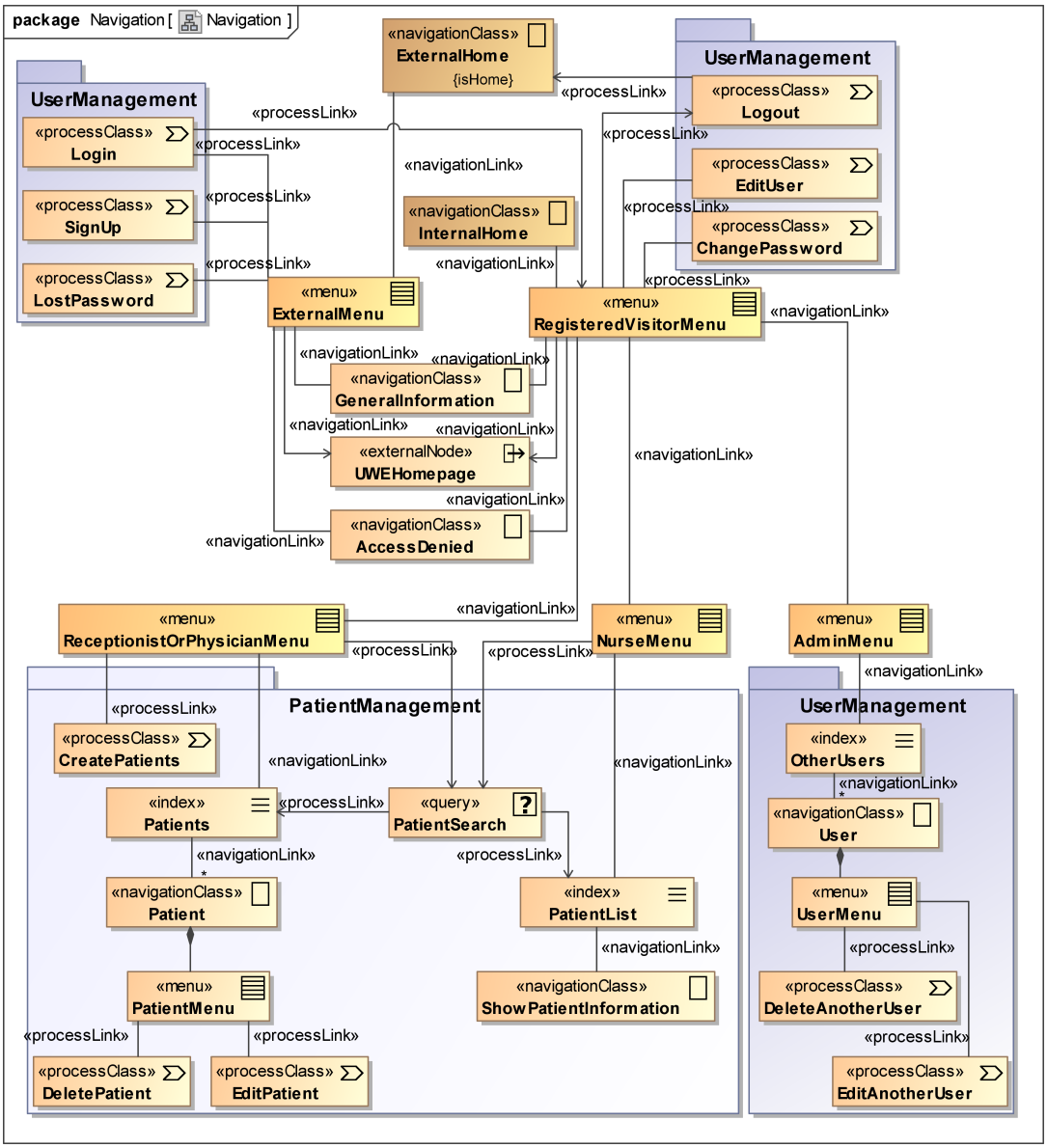


Figure 6.9: HospInfo. Navigation classes diagram

An advantage of the navigation class model is the clarity of the diagram that is supported by the division in several packages, in this case in `UserManagement` and `PatientManagement`. But as already discussed in section 4.2, security features and sessions have no representation in this traditional navigation model.

### 6.2.6 *Presentation Model*

This subsection gives an example of the abstract presentational view that is modeled with UWE, wheras the concrete layout of *HospInfo* is presented in the next chapter (section 7.2).

Figure 6.10 depicts the default page layout: a `Headline` «text» (≈) on top, a `Footer` at the bottom of the page and on the left a `NavigationMenu` with anchors (−) that determine the body of the `ContentArea`, «presentationAlternative» (⧉) class, which means that exactly one of the included «presentationGroup» (▣) properties can be displayed simultaneously.

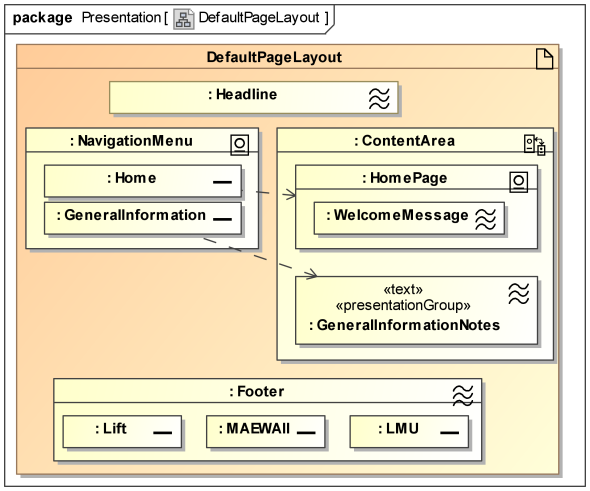


Figure 6.10: HospInfo. Presentation diagram template

The diagram in figure 6.11 shows the concrete classes for the nurse area. The classes `NavigationMenu` and `ContentArea` are the same as in figure 6.10, but different properties are modeled.

On the left of figure 6.11 the menu is depicted that can be unfolded so that `AllCategories` «presentationGroup» (▣) i.e. all submenus are displayed. Again dependencies are added to understand the effect of a link. For example the `ContentArea` shows the patients of the same ward after the according anchor in the menu had been activated.

### 6.2.7 *Process Model*

The UWE process model is used to present detailed workflows. For *HospInfo* example diagrams are introduced for the process of creating a patient and for the process of editing user properties by an administrator.

Figure 6.12 depicts the first case and the UWE stereotypes «user-Action» (🏃) and «systemAction» (▯) are used to draw a distinction between actions that are carried out by the user or by the system. Thus the user edits a new patient (`EditNewPatient`), but

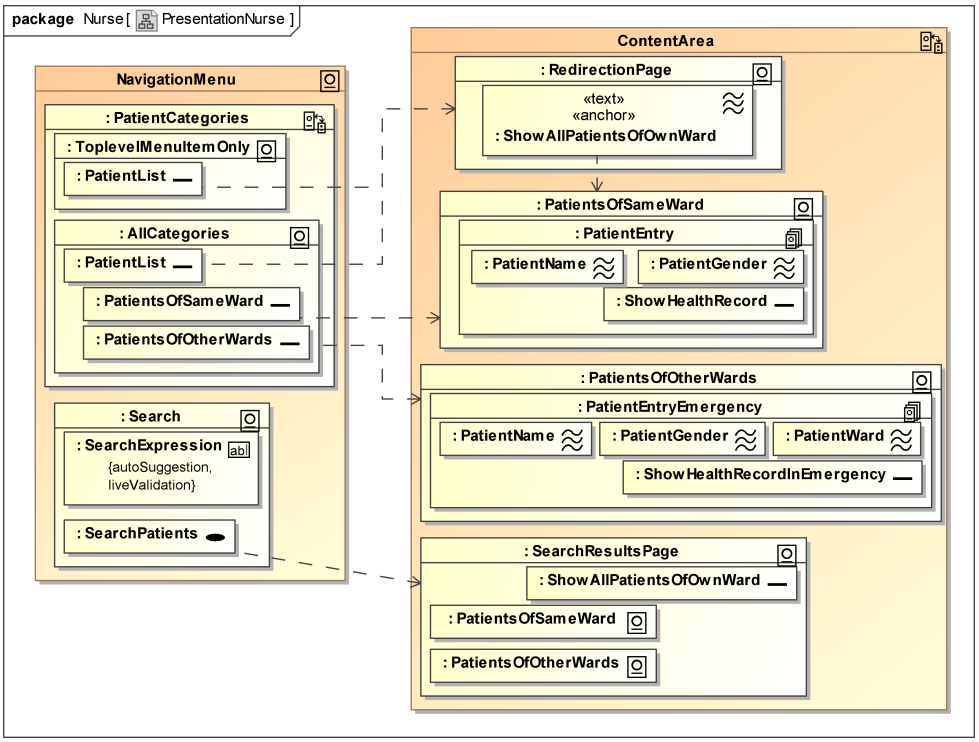


Figure 6.11: HospInfo. Presentation for nurses

the validation of the data is done by the system. It is noticeable that we do not model any distinction between system actions that are executed on the server and those which are processed by the client.

Figure 6.13 shows a similar activity diagram for the process of editing user properties. The difference to figure 6.12 is that if the administrator e.g. wants to remove the 'registeredVisitor' role for a user, *HospInfo* corrects this value and generates a hint. It is a bit unusual to implement the registered visitors as a role, because all users that are available in the system are of course 'registered', but we wanted to stick closely to the model in this case study.

Due to the lack of space, not all diagrams could be depicted in this chapter. The interested reader may refer to the attached MagicDraw project on CD.

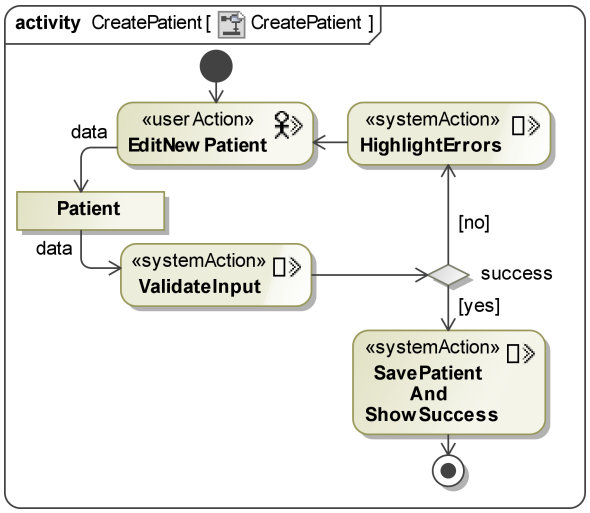


Figure 6.12: HospInfo. Create patient process

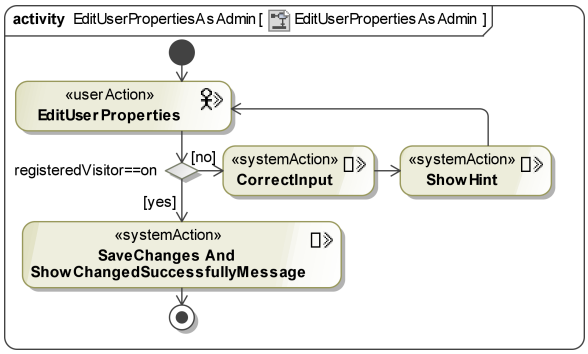


Figure 6.13: HospInfo. Edit user properties process

# 7

# CASE STUDY – IMPLEMENTATION OF HOSPINFO

The web framework Lift has been chosen for the implementation of *HospInfo*. This chapter starts with a brief introduction of Lift and the underlying language Scala. Afterwards we discuss some aspects of the implementation and finally, we sum up our lessons learned while transforming our *HospInfo* model into a running application.

## 7.1 Selection of a Web Framework

*web framework requirements*

For the implementation of *HospInfo* a web framework has to be selected from the many available ones. A tabular comparison can be found at Wikipedia.[1] For *HospInfo* an implementation of a prototype without having to configure too many components (convention over configuration) is as important as the support of security frameworks. Furthermore Ajax has to be available to enable up-to-date RIAs and the underlying programming language must be expressive and easy to learn.

*decision to choose one of the newer web frameworks*

These requirements do not really narrow the results, so we decided to gain experience in some of the newer languages and frameworks, like Ruby[2] on Rails[3], Groovy[4] / Grails[5] or Scala[6] / Lift.[7] In the end, Lift provoked curiosity, not only because it comprises many features of former frameworks, but also because it is based on Scala, a multi-paradigm programming language that allows object-oriented as well as functional programming. Another considerable possibility is Sif, a "framework for building high-assurance web applications, using language-based information-flow control to enforce security" [8]. We have decided to use Scala instead of Sif, because information flow is out of the scope of our modeling techniques so far and the Scala / Lift community is many times greater at the moment.

1 Wikipedia: Comparison of web application frameworks. http://en.wikipedia.org/wiki/Comparison_of_web_application_frameworks, last visited 2010-10-24
2 Ruby Programming Language. http://www.ruby-lang.org/, last visited 2010-10-25
3 Ruby on Rails. http://rubyonrails.org/, last visited 2010-10-25
4 Groovy. http://groovy.codehaus.org/, last visited 2010-10-25
5 Grails. http://grails.org//, last visited 2010-10-25
6 The Scala Programming Language. http://www.scala-lang.org/, last visited 2010-10-25
7 Lift web framework. http://liftweb.net/, last visited 2011-02-12

### 7.1.1 *Scala Features*

Scala is a good way to try something new and interesting in order to improve skills in functional programming without abandoning the option of imperative programming at all. Apart from that, Scala is based on Java, runs on the Java Virtual Machine (JVM) and therefore allows the programmer to use Java libraries. This feature is very useful, not only because the programmers can keep their Java code, but also because Java provides libraries of many kinds.

Examples are the libraries of *Java SE Security*.[8] They provide functionality for e.g. authentication and access control, PKI and secure communications and there is no need for Scala to reimplement these practical and well tested features.

*Scala example*

In addition, Scala's strengths are beyond Java and comprise code patterns as well as the concise style of functional programming languages. A short example for a code pattern is the iterator pattern, which is integrated in the language: [53, chapter 13]

```
"Programming Scala" foreach {c => println(c)}
```

This example iterates over each character and prints it in a new line. Even if `foreach` looks like a keyword, it is none. An implicit conversion converts the Java `String` into a Scala `RichString`, which provides a `foreach` method. Methods in Scala can be invoked by replacing the common Java-like dot before with a space and if the method only takes one argument, parentheses can be avoided or substituted with round brackets.

*learning Scala*

At a first glance, Scala takes a little time to getting used to, but with a basic knowledge of functional languages a programmer can easily learn the ropes by reading tutorials (e.g. [46] or German ones [20, 15]) and go into more detail with [53], [41] (German) or the book of the designer of Scala, Martin Odersky [38].

### 7.1.2 *Lift Features*

The Lift web framework was launched 2007 and version 2.0 was released in June 2010. We used version 2.2 for the implementation; the version 2.2-M1 (based on Scala 2.8.0) had been used until 2.2 (based on Scala 2.8.1) was finally released in January 2011. The Lift homepage claims:

> Lift applications are:
>
> - Secure – Lift apps are resistant to common vulnerabilities including many of the *OWASP Top 10*

8 Java SE Security. http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136007.html, last visited 2010-09-30

- Developer centric – Lift apps are fast to build, concise and easy to maintain
- Scalable – Lift apps are high performance and scale in the real world to handle insane traffic levels
- Interactive like a desktop app – Lift's *Comet* support is unparalled and Lift's ajax support is super-easy and very secure.[7]

*OWASP*

The *OWASP Top 10*[9] of the year 2010 contains e.g. injection flaws (SQL or others), Cross-site scripting (XSS) and broken authentication and session management. *Comet* is a model where the client sends a request to the server and the server responds to the RIA when data is available. Afterwards a new request is sent by the client to give the impression that the server is notifying the client automatically. Despite the advertising text above, Lift still is in the early stages of development, i.e. the documentation is under development and some features are supposed to be changed, e.g. the primary object-relational mapping (ORM) framework. [7, chapter 8] At the moment, [7] is the only Lift book available, but the Lift community leverages a Wiki[10] and a German reader might start with an online tutorial of Heise Developer [4].

*Comet*

*Lift documentation*

*working environment*

Scala is supported by many integrated development environments (IDEs), e.g. by Eclipse[11] (Scala-IDE[12]) and by NetBeans[13] (Scala Plugin[14]). Testing web applications can be a tedious task, as usually all sources have to be recompiled and the servlet container (e.g. Tomcat[15] or Jetty[16]) has to be restarted to load the updated files. If Eclipse is used, this process can be simplified by using Maven[17] (a build system) and JRebel[18], which is free for Scala developers. Eclipse supports the incremental build process (`Project / Build automatically`) and JRebel automatically reloads the relevant files without enforcing a complete Jetty restart. Consequently, we use Eclipse Helios with the Scala-IDE 2.8.1 (nightly), the maven plugin m2eclipse[19] together with Jetty and JRebel 3.6.1.

9 OWASP Top 10 2010. http://owasptop10.googlecode.com/files/OWASP%20Top%2010%20-%202010.pdf, last visited 2010-10-20
10 Lift Wiki. http://www.assembla.com/wiki/show/liftweb, last visited 2010-10-25
11 Eclipse. http://www.eclipse.org/, last visited 2010-10-16
12 Scala-IDE. http://www.scala-ide.org/, last visited 2011-02-12
13 NetBeans. http://netbeans.org/, last visited 2010-10-16
14 Scala Plugin. http://wiki.netbeans.org/Scala, last visited 2010-10-16
15 Apache Tomcat. http://tomcat.apache.org/, last visited 2010-09-01
16 Jetty. http://jetty.codehaus.org/jetty/, last visited 2010-09-01
17 Apache Maven. http://maven.apache.org/, last visited 2010-09-01
18 JRebel. http://sales.zeroturnaround.com/, last visited 2011-02-12 (the Scala version can be accessed directly from the 'Sales' page)
19 M2Eclipse. http://m2eclipse.sonatype.org/, last visited 2010-10-25

## 7.2 Realization based on Scala and Lift

This section describes the implementation of *HospInfo*, which is based on Scala and Lift. After brief instructions how to start with a Lift project, the parts of the application that realize the modeled security features are presented. These are: user management, authentication aspects, secure communication and logging, in particular the logging of Break-Glass Policy (BGP) actions.

*creating a Lift project*

The *HospInfo* Lift application was created and built using the commands that are introduced in the following:

```
mvn archetype:generate \
-DarchetypeGroupId=net.liftweb \
-DarchetypeArtifactId=lift-archetype-basic_2.8.1 \
-DarchetypeVersion=2.2 \
-DarchetypeRepository=http://scala-tools.org/repo-snapshots \
-DremoteRepositories=http://scala-tools.org/repo-snapshots \
-DgroupId=de.lmu.ifi.pst.uwe \
-DartifactId=his \
-Dversion=1.0
```

This Maven command creates a new default Lift 2.2 project (based on Scala 2.8.1) called 'his', as *HospInfo* is a prototype of a Hospital Information System. On windows each backslash (\) has to be replaced by a caret (^) in order to form a command with more than one line.

We want to use eclipse with the plugins m2eclipse and the Scala-IDE, therefore we change into the `his` directory and execute `mvn eclipse:eclipse` to create appropriate project files (cf. figure 7.1). Afterwards the `.project` file has to be changed so that the Scala `<nature>` is the first one. The java `<buildCommand>` can be deleted. UTF8 should be used as encoding for the Scala source files, as otherwise strange errors may arise (e.g. the compiler complains that semicolons are missing, although Scala mostly does not need any). Finally `**/*.java` has to be replaced with `**/*.java|**/*.scala` in the `.classpath` file.

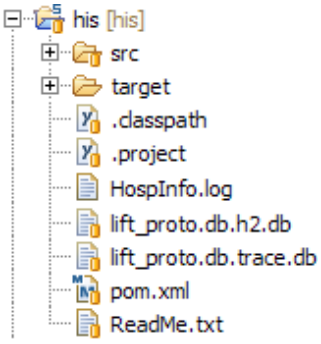


Figure 7.1: HospInfo. Excerpt of project file tree

To launch the web server jetty with *HospInfo*, the command `mvn jetty:run` is used. JRebel automatically reloads recompiled files, if the `MAVEN_OPTS` system variable is set to `-noverify -javaagent:"PathTo/jrebel.jar"`. When using JRebel, jetty's `<scanIntervalSeconds>` entry should be set to `0` in our Maven configuration file (`pom.xml`, whereas POM stands for Project Object Model) to avoid unnecessary reboot cycles.

### 7.2.1 *User Management*

The basis of the following subsections is the user management of *HospInfo*. The `model` of the basic Lift project already comprises a default user in the `User.scala` file as can be seen in the source folder in figure 7.2.

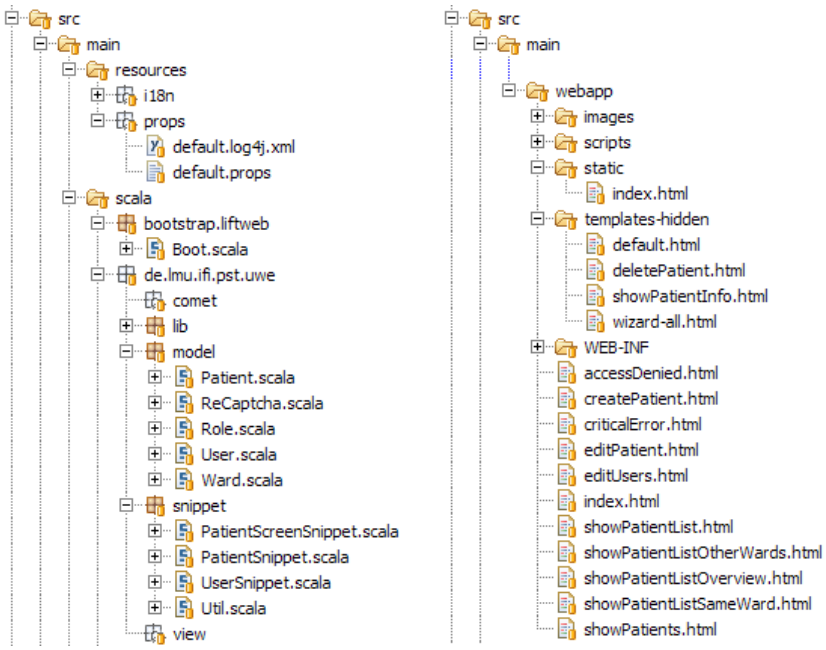


Figure 7.2: HospInfo. Excerpt of source file tree

Lift recommends a standardized package hierarchy and the content of the `model` folder corresponds to the classes and enumerations of the UWE content and user model. Therefore the Role enumeration can also be found in this model, but there are no native enumerations in Scala that is why the following code is used:

```
object Role extends Enumeration{
    val RegisteredVisitor, Admin, Physician, Receptionist,
        Nurse = Value
}
```

`Enumeration` is a built-in Scala class and `Role` an *object* (contrary to the *class* in our UWE model, cf. section 6.2.2), which means that it is not instantiable.

*interaction of user and role*

The default Lift user is enhanced by `roles` and `ward` objects, which are comparable to the association roles of our UML class `Role`. These nested objects make it possible to use the create, read, update and delete (CRUD) principle. This means the object may specify a default value or a function for verifying the value. The advantage is that other methods can use the `roles` and `ward` definitions without having to care about their scope. Addition-

ally, the ORM framework Mapper encapsulates the persistence of the user in a database (see `*.db` files in figure 7.1) and the containing objects are stored automatically as soon as they extend abstract Mapper classes, as for example `object roles extends MappedEnumList` or simpler data types as `MappedString`.

*default structure of Lift projects*

Some important Lift source folders are shown in figure 7.2: `Resources` for configuration files, `scala` for the source code and `webapp` for HTML templates, images and javascript files.

Apart from the `model`, the `scala` folder contains so called `snippets`, i.e. segments of code that are referenced from the HTML templates. The package `comet` normally includes Ajax features, but in this case it is empty because for *HospInfo* we rely on existing implementations of interactive elements, as for example the search field. Besides, the package `bootstrap.lifweb` contains the class `Boot` that constitutes the starting point of the web application.

For the user management, our `UserSnippet` provides an `edit` functionality, which is called from within the `editUsers.html` file. But before an administrator can access this page, the application has to handle authentication and access control as described in the following subsection.

### 7.2.2 *Authentication and Access Control*

A basic registration, authentication and lost password page is already included in the default Lift project. We decided to add google's reCAPTCHA service to the registration form in order to check if the visitor is a human being. For the incorporation of reCAPTCHA, an additional class is used and a reference to the library has to be added to `pom.xml`. Additionally, a translation is provided in the `i18n` folder.

After the login, there are two common ways to manage access control: On the one hand the application can use a snippet to control the access; an example is a simple check if the user is logged in or not. This is used in *HospInfo* if and only if just a small part of the page should be adapted, e.g. the home page of *HospInfo* presents the assigned roles to a logged in user and otherwise an invitation to navigate to the login form. On the other hand the general access control mechanism of Lift can be used, which is configured in the `Boot` class. The next listing shows a single line that is responsible for the decision if the nurse role exists in the current user's set of `roles`.

```
val loggedInNurse =
    If(() => (User.loggedIn_? &&
              User.currentUser.open_!.roles
                              .exists(x => x == Role.Nurse)
              ), () => RedirectResponse("/accessDenied"))
```

This line can be used for the construction of Lift's navigation menu. The menu is constructed with a list; the concerned part of menu entries is presented in the following lines. The colons connect the nurse menu with all other menus, which are taken over from our *UWEsecurity* navigation menu diagram (figure 6.6 in section 6.2.4).

```
:: Menu(Loc("Patients", List("showPatientListOverview"),
            "Patients", loggedInNurse),
   Menu(Loc("Show All Patients", List("showPatientList"),
            "Show All Patients", loggedInNurse, Hidden)),
   Menu(Loc("Same Ward", List("showPatientListSameWard"),
            "Same Ward", loggedInNurse)),
   Menu(Loc("Other Wards", List("showPatientListOtherWards"),
            "Other Wards", loggedInNurse))) ::
```

For a menu, the name of the menu entry, the location of the page and the access rights have to be specified (cf. figure 6.7). `Hidden` menus are not displayed in the navigation menu, but authorized users can navigate to the page anyway, e.g. to show all matching patients in one list after the submission of a search term, as depicted in figure 7.3 (physician view) and figure 7.5 (nurse view).

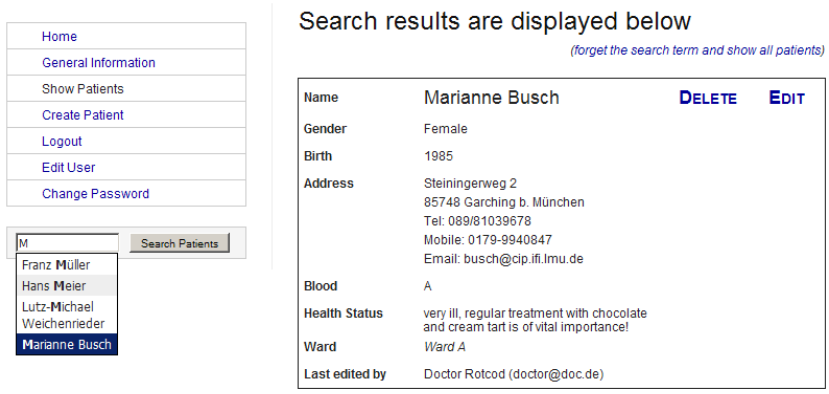


Figure 7.3: HospInfo. Patient search (physician)

After this menu configuration, Lift automatically prohibits the unauthorized navigation to a page and instead executes the given `RedirectResponse` instruction (of course any other function may be used instead).

Up to now we have implemented the modeled access on navigation nodes, but the information of the basic rights diagram is still missing. As already mentioned in the previous subsection, Lift supports CRUD, thus we have to specify proper validation methods for the nested objects in `Patient`. Lift and CRUD do not only automate the database and table creation, definition and modification, but also the GUI construction. Consequently e.g. the creation of patients is really concise: the snippet and the

HTML template code together consist of less than twenty short lines due to the fact that they just have to specify where the form should be displayed and which properties of the patient are editable. At that point the *UWEsecurity* basic rights diagram has to be taken into consideration, because receptionists are only allowed to specify values that are not health-related.

### 7.2.3 *Secure Communication*

Figure 7.4 depicts the SSL connection in the browser Firefox.[20] In the page properties it can be seen that *HospInfo* establishes a secure Advanced Encryption Standard (AES) connection with 128 bit. This is the implementation of the «session» {transmissionType='cif'} tag of our *UWEsecurity* navigation state model that is specified in section 6.2.4, figure 6.7.

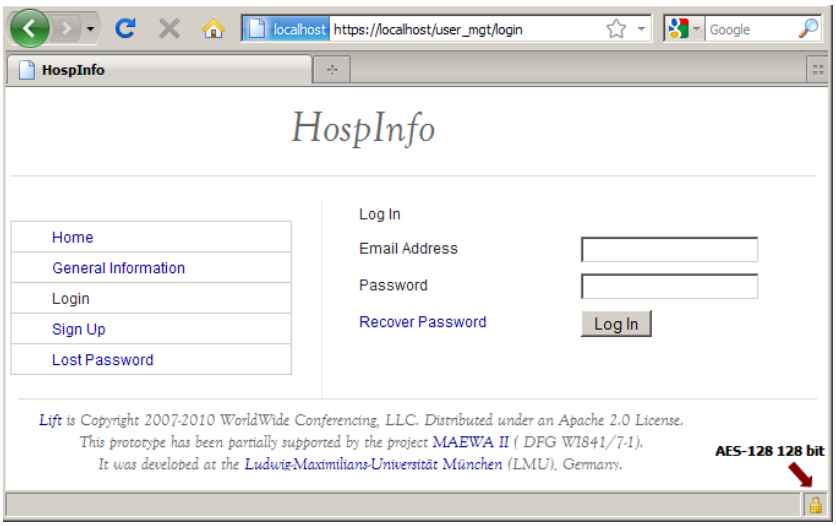


Figure 7.4: HospInfo. Login

*SSL server certificate*

To secure the communication between the server and the client with confidentiality, integrity and freshness, an SSL server certificate has to be created and signed by a CA. For our prototype we use a no-cost self-signed certificate.[21] The disadvantage is that the browser displays a warning, because the own CA is of course not included in Firefox's predefined list of root certificates.[22]

*web server configuration*

The web server Jetty has to be configured accordingly.[23] Therefore the following lines are added to the Jetty configuration in

20 Firefox 3.6 http://www.mozilla-europe.org/en/firefox/, last visited 2011-02-14

21 Ubuntu Documentation. Certificates. https://help.ubuntu.com/8.04/serverguide/C/certificates-and-security.html, last visited 2011-02-13

22 Firefox. Included Certificate List. http://www.mozilla.org/projects/security/certs/included/, last visited 2011-02-14

23 Jetty SSL Configuration. http://docs.codehaus.org/display/JETTY/How+to+configure+SSL, last visited 2011-02-13

our `pom.xml` file. Basically, the port of the new SSL connector is set to the typical https port 443 and a `keystore` file has to be created from the signed certificate.

```
<connectors>
 <connector implementation=
            "org.mortbay.jetty.security.SslSocketConnector">
   <port>443</port>
   <maxIdleTime>60000</maxIdleTime>
   <keystore>
        ${project.build.directory}/demoCA/jetty-ssl.keystore
   </keystore>
   <password>password</password>
   <keyPassword>key password</keyPassword>
 </connector>
</connectors>
```

After a restart of Jetty, *HospInfo* can be accessed at `https://localhost/`. It is notable that the registration page is the only page that does not establish a valid SSL connection, because the CAPTCHA is used as a service from google, thus the browser shows the error "connection partially encrypted" with good reason.

### 7.2.4 *Logging and Break Glass Policy*

For the implementation of the logging functionality, the log4j[24] library is used and referenced in our `pom.xml`. We particularly need the logging to realize the BGP concerning nurses. This means nurses are allowed to access patients of other wards in case of emergency, as depicted in figure 7.5, but the access is logged. In addition, all changes of electronic patient records (EPRs) are recorded as well.

*logging with log4j in Scala*

The log4j configuration file `default.log4j.xml` that is located in the `resources` folder does not only define a `ConsoleAppender` (for writing logs to the console), but also a `FileAppender` that appends all new entries to the file `HospInfo.log` in the project root (cf. figure 7.1). Logging in Scala is simple, the class or object just has to mix in the trait `Logger` or similar traits from the same library, as e.g. `Loggable`, to get a nested logger that can be accessed in the following way: `logger.info("String to log")`.

## 7.3 Lessons Learned

After the description of the *HospInfo* implementation in the beginning of this chapter, this section outlines our experiences with

24 Apache Logging. log4j. `http://logging.apache.org/log4j/`, last visited 2011-02-14

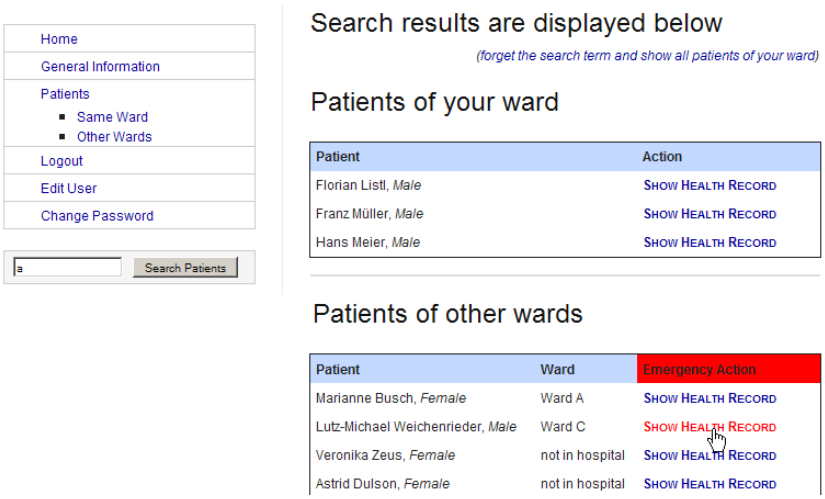


Figure 7.5: HospInfo. Patient list (nurse from ward B)

the realization of our new UWE model and with technical issues regarding the work with Scala and Lift.

*realization of the* UWEsecurity *model*

In general the realization of *UWEsecurity*'s security features is straightforward, as it is tailored for web applications. Even if the navigation state model tends to become very complex, it is useful, as the presented information cannot be derived directly from the code. Consequently, the programmer can take care of the overall navigation structure while implementing parts of the application. The navigation menus in Lift are not implemented with inheritance, but together with the «session» {roles} tags it is clear how to build *HospInfo*'s access controlled menu structure. The basic rights model can also be realized easily, because both the model and the code rely on the CRUD principle.

Only the creation of an UWE presentation model takes too much time in comparison with the CRUD approach that is supported by web frameworks like Lift. Therefore we suggest a mixture of prototyping with Lift and graphical modeling for the other UWE models. This would avoid spending about one week for modeling the UWE presentation whereas only a few days are needed for implementing it in Lift. Ideally, the basic Lift `model` files are generated from the UWE content and user model so that the GUI prototype can be developed even more quickly.

*technical aspects of Scala and Lift*

The first impression of Scala is that it is quite complex and there are many ways to express the same thing. But in the end the learning curve is not very steep, if someone is already familiar with functional and imperative programming. Scala's documentation is extensive, but the poor tool support of the Scala-IDE acts as a deterrent, because Eclipse freezes from time to time and the plugin is extremely slow. In extenuation of those errors it has to be said that the Scala-IDE for eclipse Helios is still experimental and therefore it is not surprising that the imports

cannot be organized correctly and sometimes inappropriate error messages are displayed hundredfold.

We consider Lift to be a web framework with high potential as it comprises useful features for many other web frameworks, e.g. Seaside's highly granular sessions and security and Wicket's designer-friendly templating style.[10] Nevertheless, its development is still at the very beginning, thus it is sparsely documented and the API changes so often that it sometimes is almost impossible to reuse older code from the web. In spite of that, the community is extremely helpful, especially the mailing list[25], and the programmer can learn a lot just by reading available code in Lift's git hub.[26]

25 Lift. Google Groups. http://groups.google.com/group/liftweb, last visited 2011-02-14

26 Lift Framework. GitHub. https://github.com/lift/framework, last visited 2011-02-15

# 8

# TOOL SUPPORT – MAGICUWE

Tool support is important for the usage of *UWEsecurity*. We decided to rely on the MagicDraw[1] CASE tool for modeling web applications. Other tools provide less functionality, particularly for UML2, and at the moment the UWE profile is also maintained with MagicDraw. Nevertheless, our tool concept may be adopted for other CASE tools like the open source toolkit TOPCASED[2].

MagicUWE *has been developed since 2007*

*MagicUWE*[3] [6] is a plugin for the CASE tool MagicDraw version 16.8. *MagicUWE* has been built for web engineers who work with the UWE(security) 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 like inserting UWE's stereotyped elements and copying UWE stereotypes and their tags. *UWEsecurity* is integrated in the UWE profile in Version 2.0, but for *MagicUWE* it is negligible in which profile the stereotype definitions are located.

The following sections introduce the functionality of the extensions of *MagicUWE* for *UWEsecurity*. The source code and the plugin can be found on the attached CD. In this chapter we do not want to dwell on the Java implementation with the MagicDraw so called *OpenAPI*, because the architecture of *MagicUWE* can easily be derived from the *Javadoc* documentation. For further details, the interested reader is referred to the bachelor thesis [5] (German 'Projektarbeit') of the author.

## 8.1 Support for Stereotypes and Tags

As can be seen in figure 8.1, the plugin *MagicUWE* adds a menu to the menu bar of MagicDraw, from which e.g. all kinds of new UWE diagrams can be created. If the users want to work with stereotyped UWE models in the containment tree, the new diagrams can be stored at the right position automatically, i.e. in a model with an appropriate UWE stereotype. The left of figure 8.1 shows that there are several modeling projects in one MagicDraw

1 MagicDraw. http://www.magicdraw.com/, last visited 2011-02-03
2 Topcased. http://www.topcased.org/, last visited 2011-02-20
3 The current version of *MagicUWE* can be downloaded from http://uwe.pst.ifi.lmu.de/toolMagicUWE.html (last visited 2011-02-03) and a reference for all features – not only the ones for *UWEsecurity* as described in this chapter – can also be found.

project. That causes the plugin to ask the user to which model the diagram should be assigned.

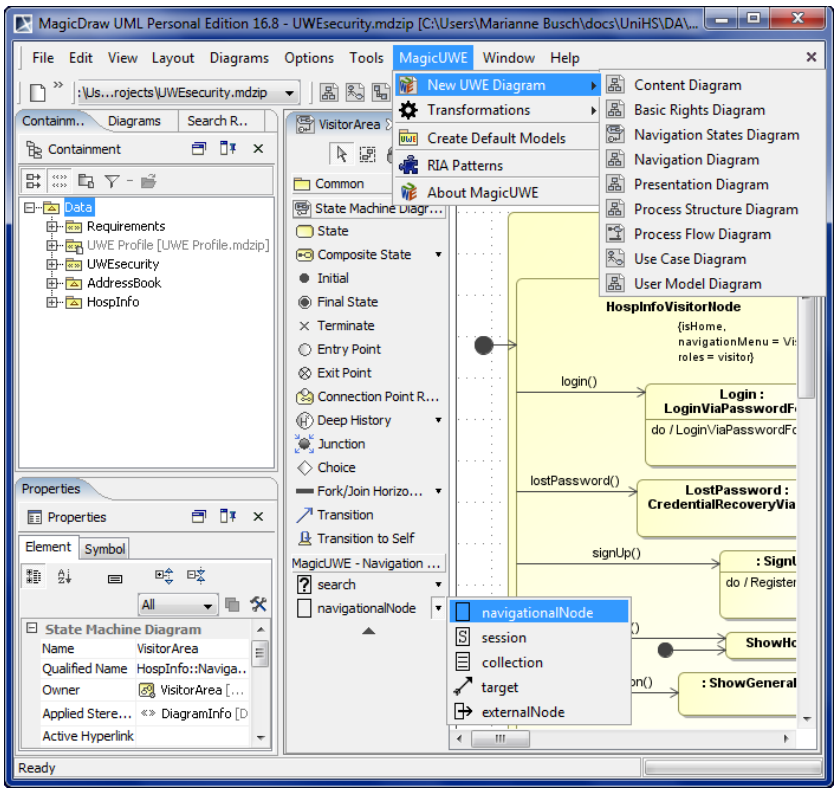


Figure 8.1: MagicUWE. Toolbar and main menus

For the developer, the most common task is to insert elements and to add stereotypes of the UWE profile. These two steps can be simplified with *MagicUWE* by choosing the stereotyped elements directly from the toolbar as depicted in the lower quarter of figure 8.1. In this case the user wants to insert a state that is typed by «navigationalNode» in a navigation state diagram. Another frequent job the specification of UWE tags, which is facilitated by a context menu (see figure 8.2).

### 8.1.1 *Easing the Use of Submachine States*

Copying stereotypes and their tags from a submachine state to the associated state machine or back is a feature which is needed due to the fact that UML allows the modeler to use a set of independent stereotypes for both, but in UWE there is no difference in the meaning. Moreover, in MagicDraw it is not possible to display tagged values in the diagram of the concerned state machine. In this way, inconsistencies may arise soon.

Figure 8.2 presents the *MagicUWE* context menu in state machine diagrams. The 'copy' submenu can only be selected on

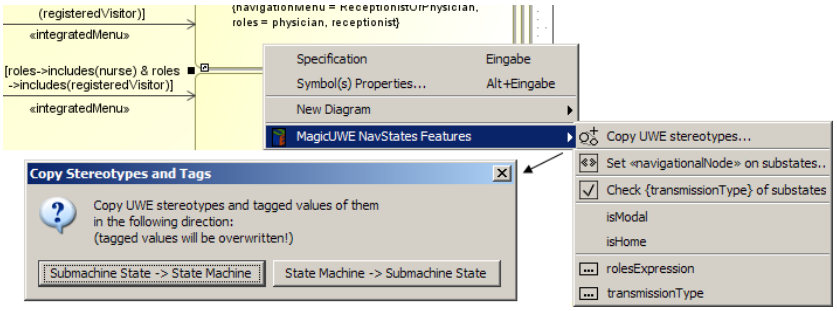


Figure 8.2: MagicUWE. Copy stereotypes from submachine state to state machine

submachine states and the user has to choose the direction. Redefined tags are overwritten, further tags remain untouched.

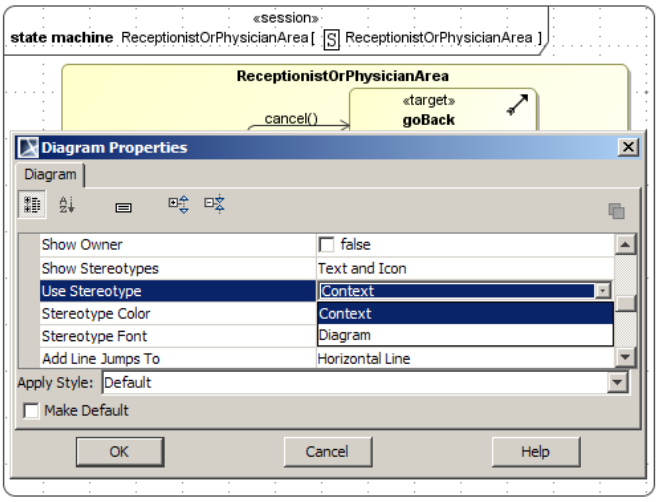


Figure 8.3: MagicUWE. Display stereotypes on state machine diagrams

By default, the stereotype of a state machine is not displayed, but that is configurable in a MagicDraw dialog. The related options can be found in the `Diagram properties` menu with the label `Use Stereotype`, where `Context` or `Diagram` can be selected for the stereotypes that should appear in the headline of the state machine diagram. The value `Context` has to be chosen for the stereotype of the state machine, as depicted at the top edge of figure 8.3.

### 8.1.2 *Default Stereotypes of Nested Elements*

Specification 3 in chapter 4 makes it possible to derive the type of a substate from the stereotypes of a superstate recursively. That means the stereotype «`navigationalNode`» is set to all substates without a stereotype that at least inherits from the «navigationalNode» stereotype. In figure 8.2 the context menu

called "`Set «navigationalNode» on substates...`" implements this transformation.

A similar case of inheritance of stereotypes is implemented for use cases stored in a package. UWE also defines the use case stereotypes on packages (cf. section 4.1) and *MagicUWE* allows the developer to apply that definition to a concrete package, so that all use cases inherit the stereotypes «browsing», «processing», «adaptation» or «observation» of the package. This is especially useful if the user plans a modification of the package structure without loosing the former stereotype information for every single use case.

## 8.2 Consistency of the Transmission Type Tag

In *UWEsecurity*'s navigation state models the transmission type can be redefined for parts of the web page and hence for parts of the navigation structure that is nested in other states. An example is the use of an insecure CAPTCHA service that is embedded on an SSL secured page.

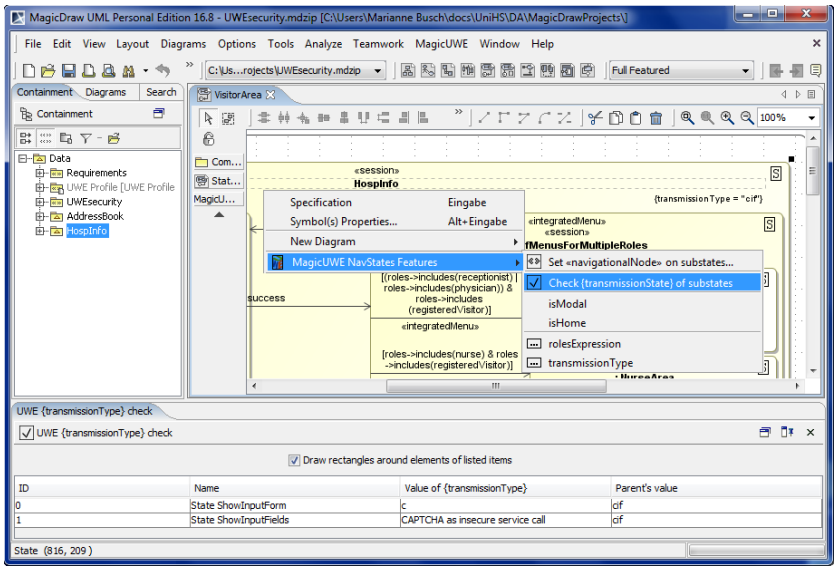


Figure 8.4: MagicUWE. Set stereotype on navigational substates, check tag {transmissionType}

*«session» ( S ) {transmissionType} redefinition check*

Figure 8.4 depicts how the consistency check functionality in *MagicUWE* can be used: the recursive check of the transmission type redefinition in substates can be executed from a context menu of a «session» state where the tag {transmissionType} is set. All results are listed in a tabbed window (see bottom of figure 8.4). Additionally red rectangles are drawn around the concerned substates in the diagrams. If the user double-clicks on a row of the result table, the element is selected in MagicDraw's containment tree.

In the example in figure 8.4, the value {transmissionType='cif'} on the `HospInfo` state is replaced in two nested substates, as e.g. in the CAPTCHA example we just mentioned. Both are located in the patterns of *MagicUWE*, therefore it would have been time-consuming for the modeler to perform this check by hand.

# Part IV

# CONCLUSION

# 9

# SUMMARY

*initial situation*

This work about the integration of security aspects in web engineering tackles the problem of insecure web applications. Earlier approaches for security engineering were too detailed and formal to be combined with web engineering. *UWEsecurity* fills this gap with a pragmatic approach.

*our solution: UWEsecurity*

Our solution consists of modeling security features graphically with UML and combining them with UWE, the UML-based Web Engineering approach. Unlike the other methods ours – called *UWEsecurity* – can also be integrated in other UML-based approaches. For web engineers it is now easy to consider security features at an early stage, without being forced to switch between two separate modeling techniques.

*security aspects*

Security aspects that are particularly important for web applications are authentication, access control, data confidentiality, data integrity and freshness. To integrate these features in the modeling process, it is useful to group them according to their granularity that is necessary for their implementation. For example, TLS connections can ensure confidentiality and freshness, therefore models for a *secure communication* link, *authentication* related processes and *Role-Based Access Control (RBAC)* are required. RBAC not only is important for creating, reading, updating and deleting objects (CRUD) and accessing their methods and attributes (as can be expressed in the basic rights model of *UWEsecurity*), but also for navigational nodes. This means some users are allowed to navigate to a particular page whereas others are not, as shown in *UWEsecurity*'s navigation state model. In this model the transmission type can also be specified, as well as in the requirements model. To avoid repetitions for common navigation structures, patterns for e.g. authentication (login), registration and password loss are provided that can easily be added as substate machine in various navigation state diagrams.

*case studies*

The applicability of the UWEsecurity approach was proven by two case studies: a Hospital Information System (HIS) and an Address Book. The first case study does not only show how to model with *UWEsecurity*, but also how modeled aspects can be realized in a concrete implementation in Lift respectively Scala. Scala is an imperative and functional programming language based on the Java virtual machine and Lift is a new web framework for Scala. The objective was to implement a prototype of a HIS that includes user management as well as security features.

Our second case study is an online address book application which allows the users to manage their address book and contacts in two partially dependent areas. Thus we are able to prove that the new navigation state model is powerful enough to express even complex navigation structures.

*tool support*

Whenever *UWEsecurity*'s graphical models are created, some tasks have to be repeated over and over. Consequently, convenient tool support for MagicDraw was created. For this purpose, the plugin *MagicUWE* was extended with features like inserting stereotyped elements, copying and automatically setting stereotypes and tags and checking the consistency of the {transmissionType} tag in substates. The next chapter presents our visions for the future of *MagicUWE* and *UWEsecurity* itself.

# 10 FUTURE WORK

Some software engineers may dream of model-to-code transformations that produce perfectly 'secure' web applications. We think this will remain a dream forever, because security has to be seen as a process rather than a feature an application might have and if a graphical model specifies every detail, it is likely to become as complex as written code.

*extensions for* UWEsecurity

Nevertheless, it is important to support this process with mature modeling techniques, which are continuously adapted to further security needs and the technical progress. An example would be UCON$_{ABC}$, a usage control system that allows for instance ongoing controls of permissions. [40] If it is used in common web applications some day, *UWEsecurity* should be enhanced accordingly.

Before thinking about new technologies, it would be interesting to try out the combination of *UWEsecurity* and other web engineering methods in practice. The UML version of WebML might be a worthwhile candidate.

A further security aspect could be the generation of a database scheme in conjunction with access rights. With some additional information, a transformation from the basic rights model would be conceivable and might be supported by *MagicUWE*. For this purpose we will consider the ongoing research work of Egea et al. (IMDEA[1], Madrid) concerning SecureUML and databases.

Even if the modeling of services has not been in the focus of UWE yet, it may be helpful to enable the specification of more details, not only concerning security features, but also general services and SOAs as an ongoing trend.

*further development of* MagicUWE

Our MagicDraw plugin *MagicUWE* has continuously been extended during the last years. Further extensions could include:

- A semi-automatic transformation between the navigation classes model and the navigation states model.
- The transformation from the basic rights model to SecureUML and a consistency check based on the {except} tags.
- Adapted transformations between the traditional UWE views so that the new views are considered as well.

In chapter 2, we have mentioned how psychological methods can boycott the hypothetically securest application and the dif-

1 IMDEA. http://www.imdea.org/, last visited 2011-02-20

ficulty in understanding a security issue and choosing the right way to tackle it. Our approach of modeling web applications before implementing them, contributes to understanding both the security problems as well as their intended solutions.

Part V

# APPENDIX

# A

# CASE STUDY – AN ADDRESS BOOK

Besides *HospInfo*, the case study of an address book has been investigated. This appendix describes the identification of requirements, the UWE content model, the user and role model, the basic rights models and its alternative representation in SecureUML, the navigation model with state machines and the presentation model for the address book case study.

## A.1 Requirements Analysis

The web application should allow registered users to create several address books and to add new contacts to one of them.

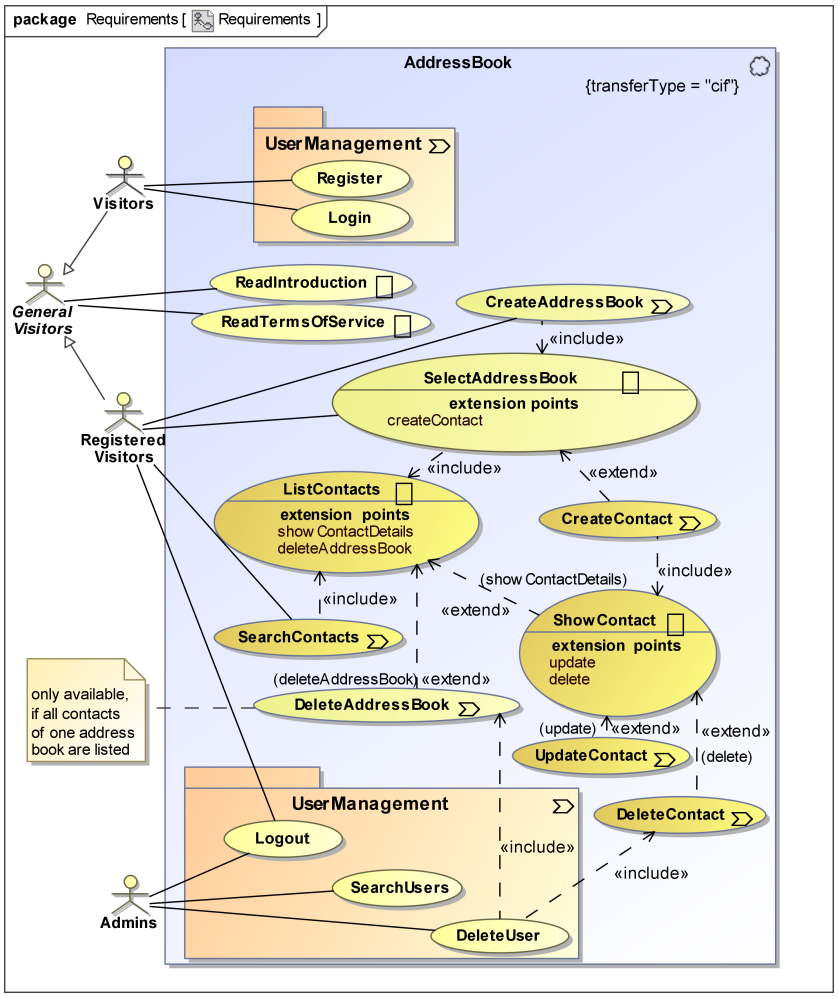


Figure A.1: Address book. Use case diagram

Non-registered visitors can only read an introduction and the terms of service until they register or authenticate themselves, as depicted in figure A.1. 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.

For registered visitors the address books are shown in a column on the left of the page. On the right the contact details of the currently selected address book are displayed. Every address book can be deleted and besides it is possible to create additional ones. The contacts can be created/removed and the user may read or update the contact details, e.g. the name, picture, postal addresses, email address or phone numbers. The latter three elements are tagged, i.e. the user can specify an arbitrary named tag for each address to distinguish between them, for example between home address and business address. Stereotypes of figure A.1 are used unspectacularly according to their definitions in section 4.1.

## A.2 Content Model

The content diagram in figure A.2 not only depicts that a user (from the UWE user model) is the owner of an unlimited number of address books, but also that each address book contains contacts with several contact details.

The abstract class `TaggedEntry` makes sure that the classes `Address`, `EMail` and `Phone` provide a `tagName` label for every object which is created by a user.

## A.3 User Model and Role Model

*user model*

In contrast to our *HospInfo* case study (section 6.2.2), a `User` is associated with exactly one role. That `Role` is modeled as a class, as introduced in the previous section, and not as an enumeration.

*role model*

In this case study we want to compare our *UWEsecurity* basic rights model (section A.4) with the SecureUML model (section A.5). Therefore figure A.3 shows the underlying *UWEsecurity* role model (cf. section 4.4.1) in the upper and the SecureUML version in the lower half of the diagram. The two versions differ in the use of linked instances versus stereotyped classes with inheritance.

## A.4 Basic Rights Model

The basic rights model comprises access rules for contacts, users and address books. In this case study we do not model the

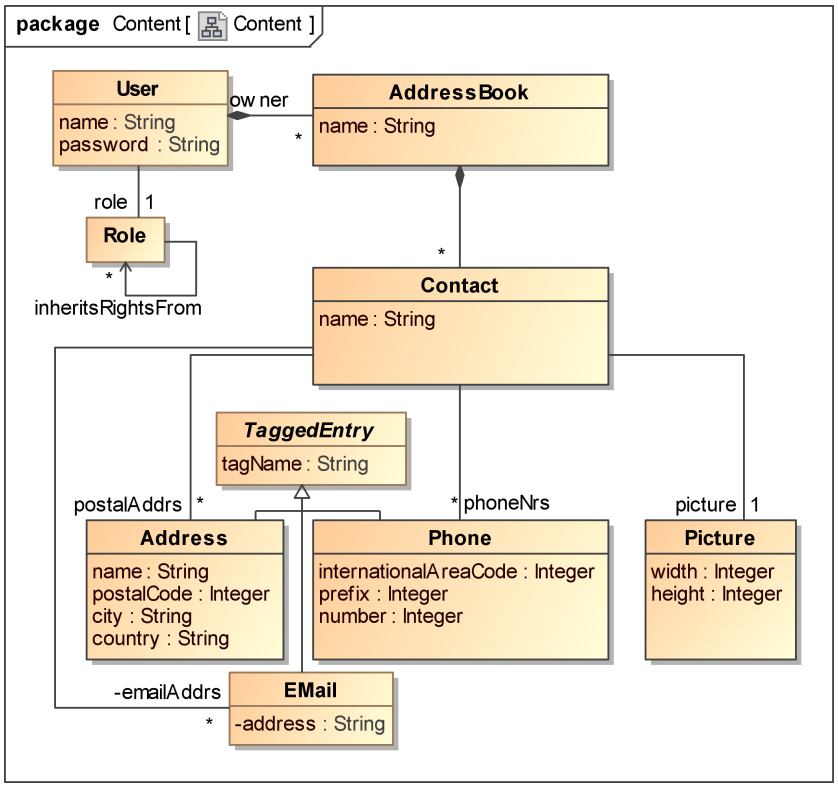


Figure A.2: Address book. Content diagram

CRUD access, but instead the execution rights on methods. All possibilities can be found in section 4.4.2.

Some comments stereotyped by «authorizationConstraint» are added in order to specify that a registered visitor is just allowed to delete his own contacts and address books. Furthermore, an administrator has the permission to delete users, except other administrators. Other constraints, as for `Contact.edit()` or `AddressBook.create()` are easily conceivable, but for the sake of clarity they are left out in this example.

## A.5 SecureUML Model

Figure A.5 shows the same facts and circumstances with a SecureUML diagram. It is noticeable that even this simple diagram looks overfilled, because of the association classes.

*comparison of the Basic Rights and the SecureUML model*

As already mentioned in section 3.1.2, SecureUML diagrams specify the permissions by repeating the (method) names of the classes in the «Permission» association classes, while the return type defines the allowed actions. The result is that – at the first glance – it is impossible to see which methods are constrained, whereas in the basic rights diagram dependencies directly point to the restricted methods in the majority of cases.

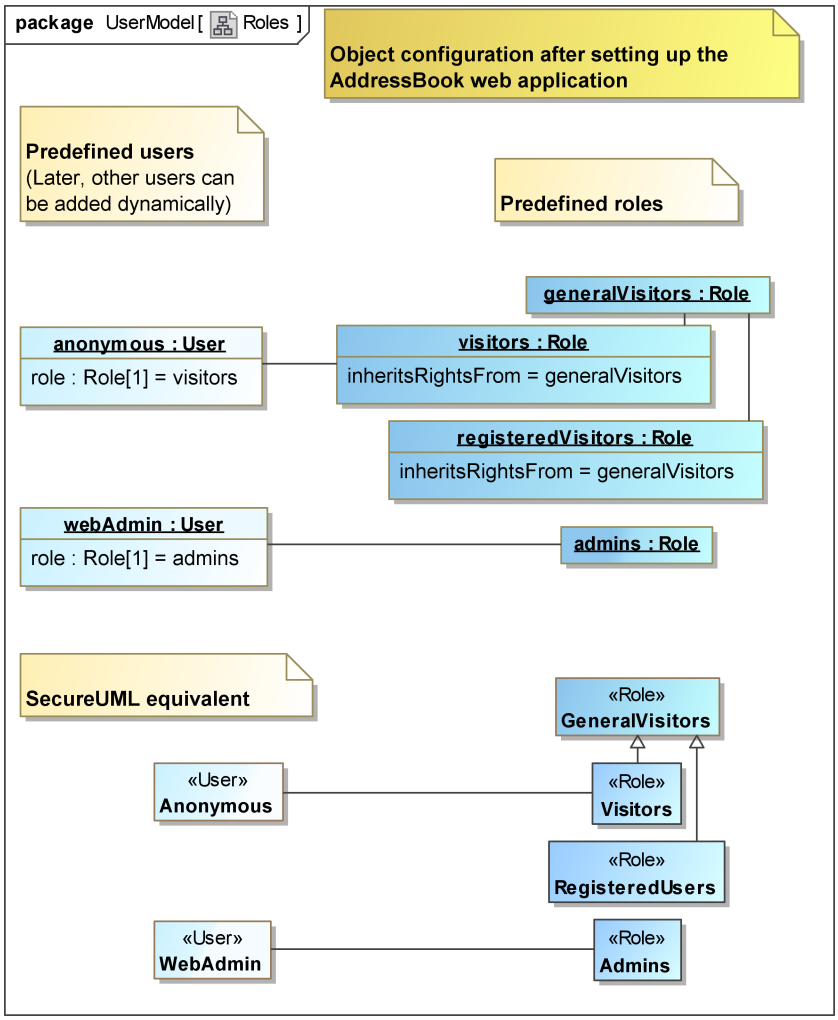


Figure A.3: Address book. Role model

If some methods of a class are executable and some not, all executable ones have to be listed separately in SecureUML. This is the reason for avoiding SecureUML diagrams in the *HospInfo* case study.

## A.6 Navigation States Model

*navigation menu*

Figure A.6 depicts the navigation menu diagram of our address book application. The menu items are modeled as methods within «navigationMenu» ( ) classes. As specified in the requirements (section A.1), the terms of service and the introduction links in the menu can be accessed by everyone.

*external area*

The «session» ( ) `ExternalArea` in the navigation states diagram (figure A.7) is denoted by the following tags, according to the specifications in section 4.2.1:

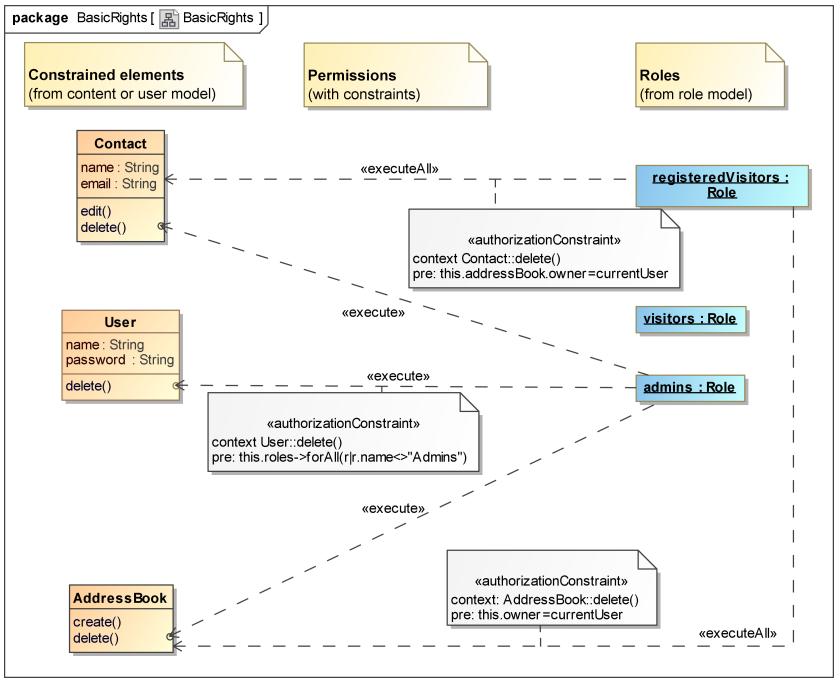


Figure A.4: Address book. Basic rights diagram

- {isHome} indicates that this state is the starting point of our web application.
- {navigationMenu=ExternalMainMenu} connects the menu class `ExternalMainMenu` with this state, i.e. the external menus are available in this navigational node.
- {roles=visitors} defines that only user instances which are linked with the `visitors` instance in the role model (see section A.3) are allowed to access this state. In this case every external visitor automatically takes on the visitor role.

After the registration and login – that are both modeled with *UWEsecurity* patterns – two types of internal areas can be reached: One for the administrators and a separated one for the registered users that want to manage their contacts. Therefore guards on transitions targeting the internal areas check the access rights. Nevertheless, the internal sessions are also labeled with {roles} tags in order to prohibit unauthorized direct access via URL.

*internal area for registered visitors*

In figure A.8 the internal area for registered visitors is depicted. The challenge is to model our two semi-independent navigation areas (address books and contacts) correctly. For that reason the orthogonal state `InternalArea` contains three regions:

The first is the `DependentArea` with the navigation for the contact area. The second region comprises only one state for the independent presentation of the list of address books. It is stereotyped by «collection» ( ) with the tagged value {item-

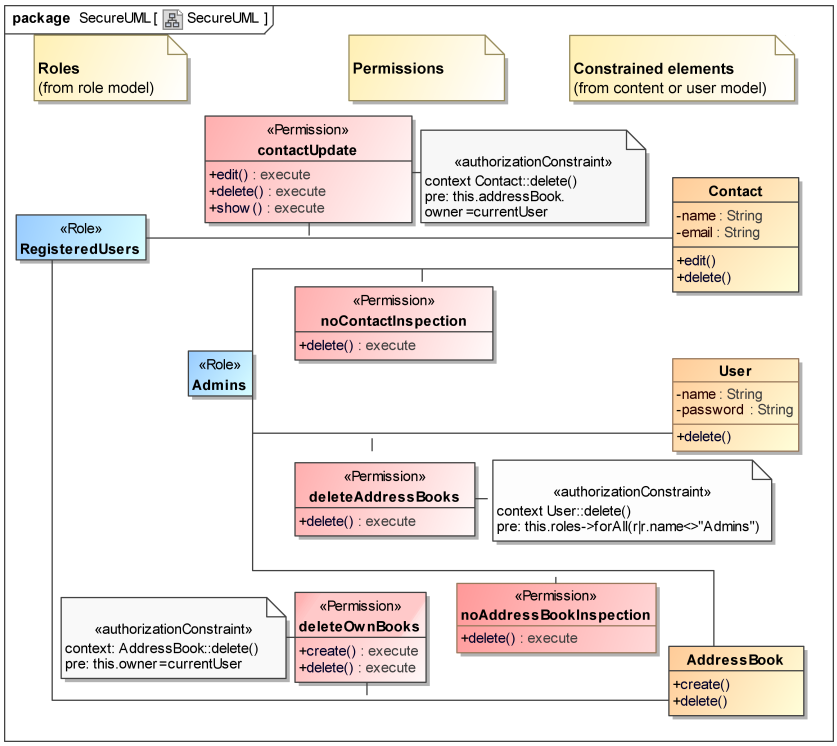


Figure A.5: Address book. SecureUML diagram

Type=AddressBook} that points to the `AddressBook` class in our content model (cf. section A.2). The third region is required for the navigation to the `CreateAddressBook` navigational node, as the creation of address books is self-sufficient.

The modal dialog for the `TermsOfService` is modeled equally (with «navigationalNode» ( ) {isModal}) as in the external area (cf. figure A.7). Even though it has to be kept in mind that both dialogs are represented by two different states, even if they have the same name.

`InnerContactArea` is a state that is nested in the `DependentArea` in order to separate the navigation for the presentation of search results and the navigation possibilities that are available after the user has selected an address book. The difference is that after a search was executed no contacts can be created and address books cannot be deleted, because the listed contacts could be located in different address books. Furthermore, the return from the `DisplayContact` submachine state is different, as in the outer state the search is executed again to update the resulting contact list. The «search» ( ) stereotype allows the modeler to replace `ct:String` with an underscore (_), but for the sake of clarity this abbreviation is not used in figure A.8.

Due to the lack of space, not all diagrams could be depicted in this chapter, but all diagrams are stored on the attached CD.

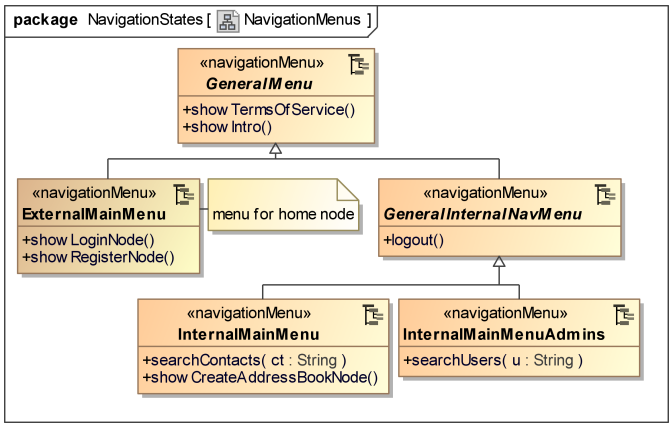


Figure A.6: Address book. Navigation Menu

## A.7 Presentation Model

As described in section A.1, the address book homepage is divided into two parts after a registered visitor has logged in: The address books are shown on the left – `AddressBooksArea`, stereotyped by «presentationGroup» ( ) – and if an address book is selected, the contacts are presented on the right (`ContactsArea`), as depicted in figure A.9. Furthermore there is a menu on top of the page, which includes links to the introduction page, to the terms of service pop-up and to a logout functionality, denoted by the stereotype «anchor» ( ).

The button `DeleteBook` is hidden if contacts from several address books are displayed after the user has executed the search. In order to keep the presentation simple, this is not modeled here, but the related navigation structure is depicted in figure A.8.

*the contacts area*

On the right, a list of contact names (`ContactListEntry [*]`), or the contact details of exactly one selected contact are shown. This exclusiveness is specified by the stereotype «presentationAlternatives» ( ). To change between both views, the anchors `ContactListEntry` and `Back` are used. Contact details that can be tagged consist of a `TagName` and a `TaggedEntry`, in which the actual contact data is shown.

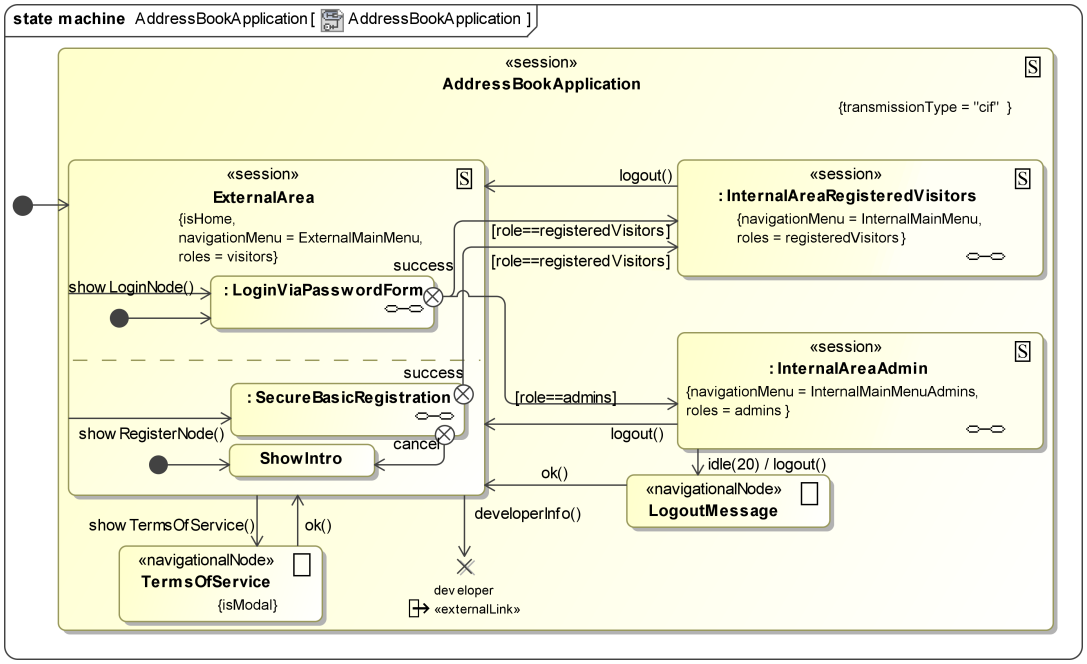


Figure A.7: Address book. Navigation of address book

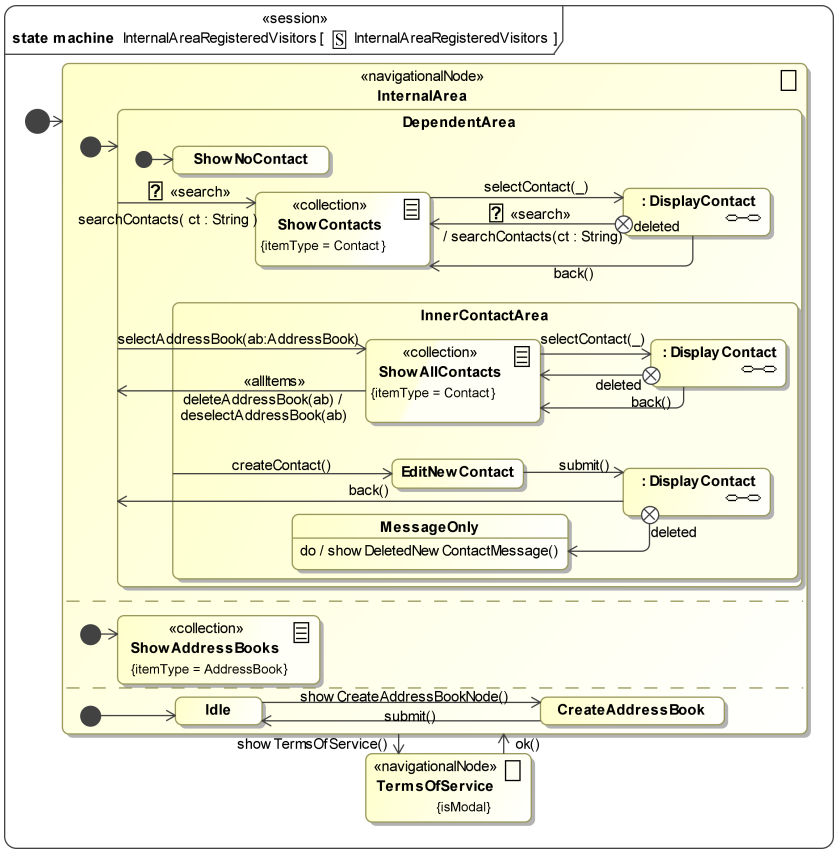


Figure A.8: Address book. Navigation of internal registered visitors

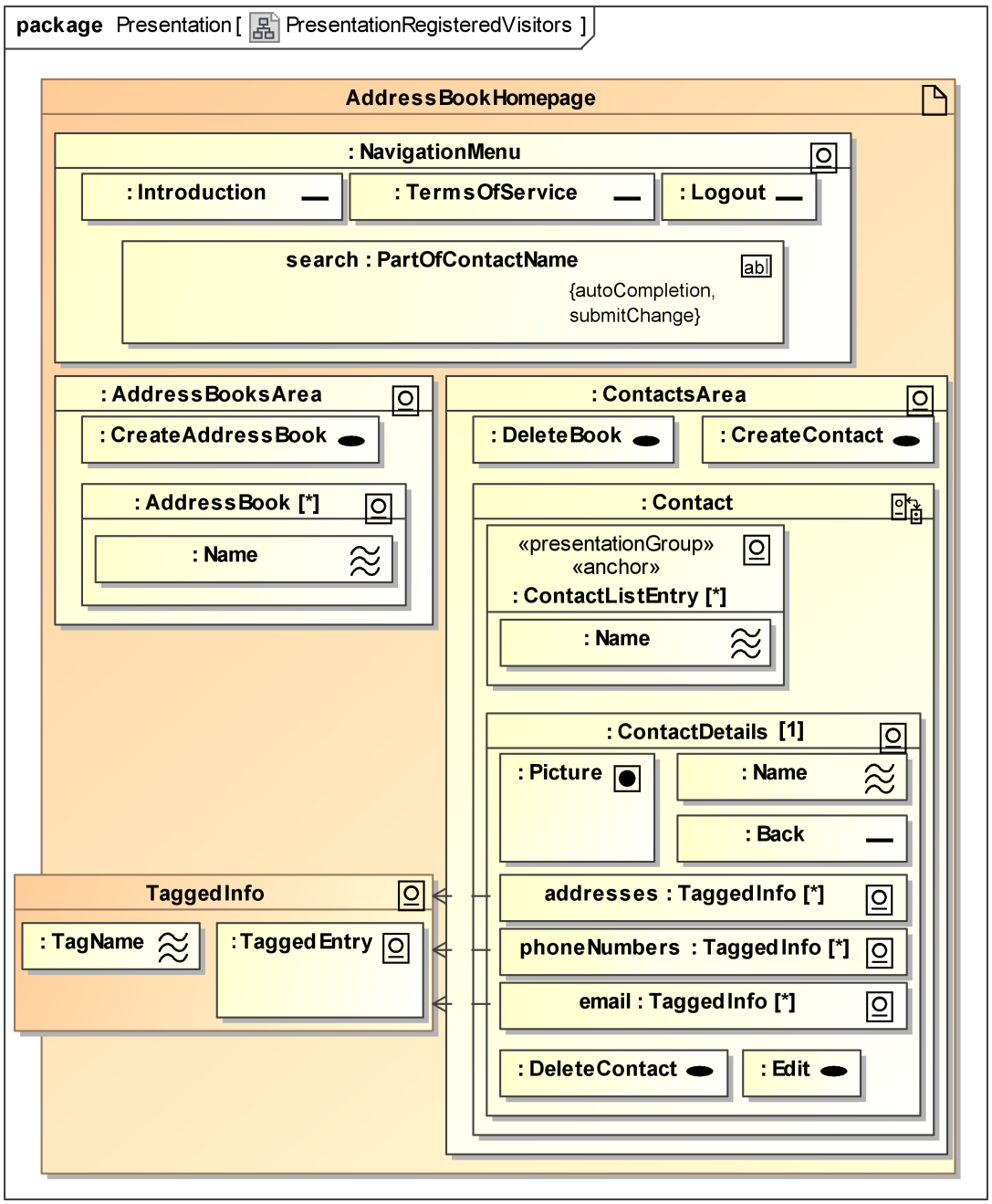


Figure A.9: Address book. Presentation

# B

## CONTENT OF THE ENCLOSED CD

The content of the enclosed CD is organized as follows:

- `/`
  - `his` ..... Implementation of HospInfo with Scala/Lift
  - `MagicDrawProjects` .... UWEsecurity profile, HospInfo and AddressBook case study
  - `MagicUWE` ..... MagicUWE version 1.3.4, Java project and plugin installers
  - `Paper` ..... Copies of the related work that is referenced in the thesis
  - `Thesis` ..... The written thesis in LaTeX and pdf format
    - `Chapters` ..... Chapters as tex-files
    - `FrontBackmatter` ... Front and backmatter as tex-files
    - `Images` ..... Images used in the thesis
    - `Presentation` ..... The *Oberseminar* presentation

# BIBLIOGRAPHY


[1] Ignacio Aedo, Paloma Díaz, and Susana Montero. A methodological approach for hypermedia security modeling. *Information and Software Technology*, 45(5):229–239, April 2003.

[2] Ross J. Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems*. Wiley, 2 edition, April 2008. ISBN 9780470068526.

[3] David Basin, Manuel Clavel, and Marina Egea. Automatic Generation of Smart , Security-Aware GUI Models. In *Engineering Secure Software and Systems*, volume 5965 of *Lecture Notes in Computer Science*, pages 201–217. Springer, 2010.

[4] Guy Philipp Bollbach and Christian Dietrich. Lift – Vehikel zum nächsten Web-Framework-Level? *Heise Developer*, September 2009. http://goo.gl/0hbxE.

[5] Marianne Busch. Migration und Erweiterung des MagicDraw-Plugins MagicUWE zur Entwicklung von Web-Anwendungen. Projektarbeit an der Ludwig-Maximilians-Universität München, 2009.

[6] Marianne Busch and Nora Koch. MagicUWE – A CASE Tool Plugin for Modeling Web Applications. In Martin Gaedke, Michael Grossniklaus, and Oscar Díaz, editors, *Web Engineering*, volume 5648 of *Lecture Notes in Computer Science*, pages 505–508. Springer Berlin / Heidelberg, 2009.

[7] Derek Chen-Becker, Tyler Weir, and Marius Danciu. *Exploring Lift*. Open Source, 2 edition, August 2010. http://groups.google.com/group/the-lift-book.

[8] Stephen Chong, Krishnaprasad Vikram, and Andrew 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, Berkeley, CA, USA, 2007. USENIX Association. http://www.cs.cornell.edu/jif/sif/.

[9] Torsten Lodderstedt David Basin, Jürgen Doser. Model Driven Security: from UML Models to Access Control Infrastructures. Technical report, ETH Zürich, July 2003.

[10] Chad Dougherty, Kirk Sayre, Robert C. Seacord, David Svoboda, and Kazuya Togashi. Secure Design Patterns. Technical report, Carnegie Mellon University. Software Engineering Institute, October 2009. http://www.sei.cmu.edu/reports/09tr010.pdf.

[11] Benjamin Fabian, Seda Gürses, Maritta Heisel, Thomas Santen, and Holger Schmidt. A comparison of security requirements engineering methods. *Requirements Engineering*, 15(1): 7–40, November 2009.

[12] Ko Fujimura and Yoshiaki Nakajima. General-purpose digital ticket framework. In *WOEC'98: Proceedings of the 3rd conference on USENIX Workshop on Electronic Commerce*, pages 15–15, Berkeley, CA, USA, 1998. USENIX Association. http://www.usenix.org/events/ec98/full_papers/fujimura/fujimura_html/fujimura.html.

[13] Stephen Gilmore, László Gönczy, Nora Koch, Philip Mayer, Mirco Tribastone, and Dániel Varró. Non-functional Properties in the Model-Driven Development of Service-Oriented Systems. *Journal of Software and Sytems Modeling (SoSym*, 2010.

[14] Network Working Group. *The Transport Layer Security (TLS) Protocol. Version 1.2*, August 2008. http://tools.ietf.org/html/rfc5246.

[15] Arno Haase. Große Oper. Scala – funktionale und objektorientierte Programmierung als pragmatische Kombination. *IX special – Programmieren heute*, 1:53–55, 2010.

[16] Munawar Hafiz. A collection of privacy design patterns. In *Proceedings of the 2006 conference on Pattern languages of programs – PLoP '06*, New York, New York, USA, 2006. ACM Press. ISBN 9781605583723.

[17] Munawar Hafiz, Paul Adamczyk, and Ralph E. Johnson. Organizing Security Patterns. *IEEE Software*, 24(4):52–60, July 2007.

[18] Michael Hafner and Ruth Breu. *Security Engineering for Service-Oriented Architectures*. Springer, November 2008. ISBN 9783540795384.

[19] Michael Hafner, Mukhtiar Memon, and Muhammad Alam. Modeling and Enforcing Advanced Access Control Policies in Healthcare Systems with S ECTET. *Elements*, pages 132–144, 2008.

[20] Christian Helmbold. Zwei Welten – Funktional und objektorientiert programmieren mit Scala. *c't – Magazin für Computer Technik*, 21:128–187, December 2009.

[21] Martin Hitz, Gerti Kappel, Elisabeth Kapsammer, and Werner Retschitzegger. *UML @ Work. Objektorientierte Modellierung mit UML 2*. Dpunkt Verlag, 3., aktualis. und überarb. a. edition, July 2005. ISBN 9783898642613.

[22] Christian Johner and Peter Haas. *Praxishandbuch IT im Gesundheitswesen. Erfolgreich einführen, entwickeln, anwenden und betreiben*. Hanser Fachbuch, March 2009. ISBN 9783446415560.

[23] Jan Jürjens. *Secure Systems Development with UML*. Springer, November 2004. ISBN 9783540007012. Tools and further information: http://www.umlsec.de/.

[24] Serge Kater. System security requirements verification using UMLsec. Master's thesis, Open University, March 2010. http://jan.jurjens.de/umlsectool/M801_Dissertation_S_Kater_X0321151.pdf.

[25] Darnell Kienzle, Matthew Elder, David Tyree, and James Edwards-Hewitt. Security Patterns Repository. *Security*, 2002. http://www.scrypt.net/~celer/securitypatterns/.

[26] Alexander Knapp and Gefei Zhang. Model transformations for integrating and validating web application models. In Heinrich C. Mayr and Ruth Breu, editors, *Proceedings of Modellierung*, volume P-82, pages 115–128. Gesellschaft für Informatik, 2006.

[27] Sergej Kozuruba. Modellbasierte Anforderungsanalyse für die Entwicklung von adaptiven RIAs. Master's thesis, Ludwig-Maximilians-Universität München, September 2010.

[28] Christian Kroiß. Modellbasierte Generierung von Web-Anwendungen mit UWE. Master's thesis, Ludwig-Maximilians-Universität München, 2008. http://uwe.pst.ifi.lmu.de/publications/christian_kroiss_Ausarbeitung_DA_final.pdf.

[29] Thomas M. Lehmann. *Handbuch der medizinischen Informatik*. Hanser Fachbuchverlag, 2 edition, December 2004. ISBN 9783446227019.

[30] Torsten Lodderstedt, David Basin, and Jürgen Doser. SecureUML : A UML-Based Modeling Language for Model-Driven Security. In *Proceedings of 5th International Conference on the Unified Modeling Language, 2002. – UML'02*, volume 2460 of *Lecture Notes in Computer Science*, pages 426–441, Berlin, Germany, 2002. Springer Verlag.

[31] Mukhtiar Memon, Michael Hafner, and Ruth Breu. SECTISSIMO : A Platform-independent Framework for Security Services. *Components*, October 2008. http://www.comp.lancs.ac.uk/modsec/papers/modsec08_submission_2.pdf.

[32] Michael Menzel and Christoph Meinel. A Security Metamodel for Service-Oriented Architectures. *2009 IEEE International Conference on Services Computing*, pages 251–259, September 2009.

[33] Nathalie Moreno, Piero Fraternali, and Antonio Vallecillo. WebML modelling in UML. *IET Software*, 1(3):67, 2007. ISSN 17518806.

[34] Haralambos Mouratidis, Jan Jürjens, and Jorge Fox. Towards a Comprehensive Framework for Secure Systems Development. *Advanced Information Systems Engineering, Lecture Notes in Computer Science*, pages 48 – 62, 2006.

[35] San Murugesan, Yogesh Deshpande, Steve Hansen, and Athula Ginige. Web engineering: a new discipline for development of web-based systems. In San Murugesan and Yogesh Deshpande, editors, *Web Engineering*, volume 2016 of *Lecture Notes in Computer Science*, pages 3–13. Springer Berlin / Heidelberg, 2001.

[36] Ernst Oberortner, Martin Vasko, and Schahram Dustdar. Towards Modeling Role-Based Pageflow Definitions within Web Applications. In *Proceedings of the 4th International Workshop on Model-Driven Web Engineering*, pages 1–15, 2008.

[37] *OMG Unified Modeling Language (OMG UML), Superstructure*. Object Management Group (OMG), 2.3 edition, 2010. http://www.uml.org/.

[38] Martin Odersky, Lex Spoon, and Bill Venners. *Programming in Scala: A Comprehensive Step-by-step Guide*. Artima Inc, November 2008. ISBN 9780981531601.

[39] Rolf Oppliger. *SSL and TLS: Theory and Practice (Information Security and Privacy)*. Artech House Publishers, September 2009. ISBN 9781596934474.

[40] Jaehong Park and Ravi Sandhu. The UCONABC usage control model. *ACM Trans. Inf. Syst. Secur.*, 7:128–174, February 2004.

[41] Lothar Piepmeyer. *Grundkurs funktionale Programmierung mit Scala*. Hanser Fachbuchverlag, June 2010. ISBN 9783446420922.

[42] Dhanya Pramod. Modeling security aspects in model driven web application. *Journal of Computer Science*, 1(1), 2008.

[43] Dhanya Pramod and Vinay Vaidya. A Platform Specific UML model for Web application self defense through an Aspect Oriented Approach. *International Journal*, 1(4):449–457, 2009.

[44] Ivan Ristic. *Apache Security*. O'Reilly Media, March 2005. ISBN 9780596007249.

[45] Sasha Romanosky, Alessandro Acquisti, Jason Hong, Lorrie Faith Cranor, and Batya Friedman. Privacy patterns for online interactions. In *Proceedings of the 2006 conference on*

*Pattern languages of programs – PLoP '06*, New York, New York, USA, 2006. ACM Press. ISBN 9781605583723.

[46] Michel Schinz. *A Scala Tutorial*, 2010. http://www.scala-lang.org/docu/files/ScalaTutorial.pdf.

[47] Jürgen Schmidt. Hash cracked. The consequences of the successful attacks on SHA-1. *Heise Security*, August 2006. http://www.h-online.com/security/features/Hash-cracked-747181.html.

[48] Jürgen Schmidt. Hashes revisited. The consequences of the successful MD5 attacks. *Heise Security*, January 2009. http://www.h-online.com/security/features/Consequences-of-the-successful-MD5-attacks-746221.html.

[49] Markus Schumacher. *Security Engineering with Patterns: Origins, Theoretical Models, and New Applications (Lecture Notes in Computer Science)*. Springer, September 2003. ISBN 9783540407317.

[50] Markus Schumacher, Eduardo Fernandez-Buglioni, Duane Hybertson, Frank Buschmann, and Peter Sommerlad. *Security Patterns: Integrating Security and Systems Engineering (Wiley Software Patterns Series)*. Wiley, March 2006. ISBN 9780470858844.

[51] Stephen A. Thomas. *SSL and TLS Essentials: Securing the Web*. Wiley, February 2000. ISBN 9780471383543.

[52] Christian Ullenboom. *Java ist auch eine Insel*. Galileo Press GmbH, January 2009. ISBN 9783836213714. http://openbook.galileocomputing.de/javainsel8/.

[53] Dean Wampler and Alex Payne. *Programming Scala: Scalability = Functional Programming + Objects*. O'Reilly Media, September 2009. ISBN 9780596155957. http://programming-scala.labs.oreilly.com/.

[54] Gefei Zhang, Nora Koch, and Alexander Knapp. Aspect-Oriented Modeling of Access Control in Web Applications. In *6th Workshop on Aspect Oriented Modeling (AOM)*, 2005.