Towards the Integration of Security Aspects into System Development using Collaboration-Oriented Models

Linda Ariani Gunawan, Peter Herrmann, and Frank Alexander Kraemer

Department of Telematics
Norwegian University of Science and Technology (NTNU)
Trondheim, Norway
{gunawan,herrmann,kraemer}@item.ntnu.no

Abstract. Security is an important feature of system design which should be taken into account early in the development of systems. We propose an extension of the SPACE engineering method in order to integrate security aspects into the system design and implementation. The integration of security mechanisms is facilitated by collaboration-oriented models of the functional system specification (i.e., by describing functionalities reaching over different physical components in one model). Countermeasures are also modeled by collaborations since security mechanisms are often collaborative structures themselves. Our approach includes an asset-oriented security analysis on the collaboration-oriented models in order to determine the level of protection needed. We illustrate our approach by the example of an e-sale system.

1 Introduction

Developing security-aware distributed systems is a non-trivial task: Not only we must guarantee the correct execution of a system provided by cooperating distributed entities with respect to the desired functionalities of the application. We also have to take into account the protection of the system against security attacks by malicious entities. The consideration of these security aspects should be integrated as early as possible into the development process, since adding security later in an ad-hoc manner increases the probability to overlook vulnerabilities contained in the system implementation [1]. For that reason, we propose a model-driven, security-aware development method based on collaborative, reusable building blocks. Application logic as well as security mechanisms are expressed by means of functionally complete UML models that can be analyzed in separation. Once consistent and adequately protected, these models can be stored as building blocks in a library. This not only reduces the development time, since proven solutions can be reused, but also facilitates the cooperation between application developers and security experts, since both parties contribute their knowledge in the form of self-contained, encapsulated units that are easy to combine, as shown in [2] for the domain of trusted systems.

Our proposed method is an extension of the engineering method SPACE [3–5] that supports the design and implementation of reactive systems in general. In this method, system specifications are expressed in the form of UML 2.x activities, enabling the composition of systems by reusable building blocks. By means of model transformation and code generation, the composed system specifications can be implemented automatically, so that engineers only have to work on the level of UML activities. We extend the SPACE method by performing a series of iterated steps of security analysis which adheres to the standard ISO/IEC Common Criteria [6]. During the analysis, security threats, attacks and risks are considered in order to design suitable protection mechanisms and integrate them into the functional specification of the system.

To be effective, most security mechanisms require the correct and coordinated collaboration of several entities. For instance, an asynchronous encryption mechanism consists at least of a transmitter encrypting a message with the public key of the receiver and the receiver itself, that decrypts the message with the corresponding private key. Furthermore, entities like a certification authority issuing certificates that a public key really belongs to a certain party are part of the security mechanism as well. For that reason, we also use collaboration models to specify countermeasures. The different stakeholders of security mechanisms and their corresponding behaviors may be expressed by UML activities which can easily be combined with other collaboration models describing the systems to be protected.

2 The Basic Development Method

The structure of the e-sale system is modeled using collaboration diagram, as shown in Fig. 1. It consists of two collaboration roles, *customer* and *merchant*, which denote the participants of the system. To make a purchase, these roles have to collaborate with each other, i.e., execute some joint behavior. This behavior is captured by the two collaboration uses *o:Order* and *p:Payment*, denoted by ellipses. The dashed lines are the role bindings specifying that the customer is the buyer during the order, and the payer during the payment. Vice versa, the merchant acts as a seller and a payee, respectively.

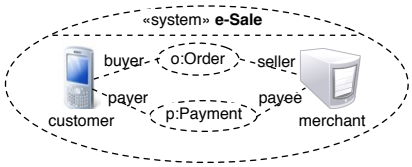

Fig. 1. UML collaboration for the e-Sale system

While the collaboration in Fig. 1 specifies from which services the complete system is composed, the detailed dependencies between order and payment are

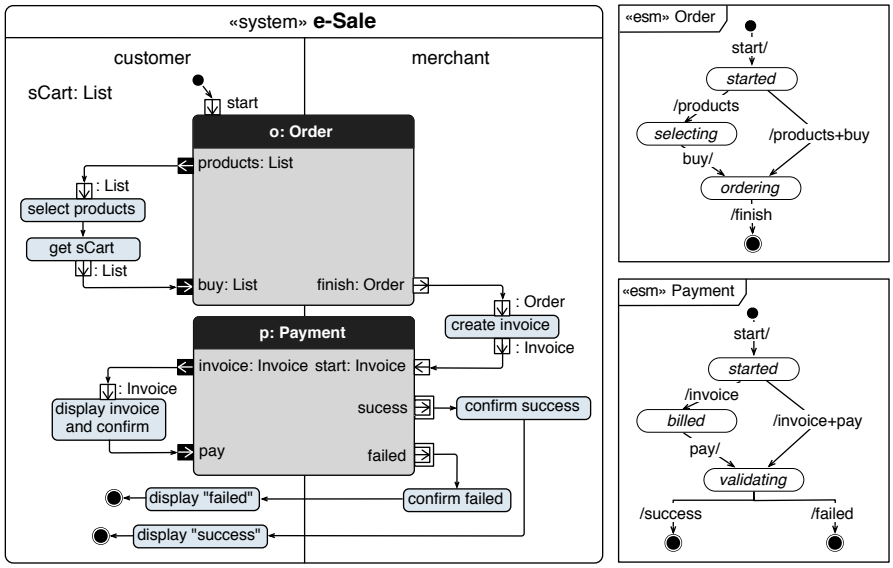

Fig. 2. The behavior of the e-Sale system

not visible. For this purpose, we use the UML activity diagram shown in Fig. 2. Each participant in the system is represented by its own activity partition.The order and payment services from Fig. 1 are represented by the call behavior actions *o:Order* and *p:Payment*, referring to activity diagrams that define their detailed internal behavior, as we will see later.

In contrast to the collaboration uses from Fig. 1, call behavior actions have pins at their borders which denote specific events that we can use for their composition. In order to understand the high-level interface behavior of the call behavior actions without looking into their internals, they are accompanied by special, external state machines (ESMs), shown to the right in Fig. 2. They define the externally observable behavior at the pins of a call behavior action.

The e-sale system starts at the initial node in the customer side by triggering *o:Order* via its start pin. From the ESM for the order collaboration shown at the upper right in Fig. 2, we see that after a start, we have to be prepared for the arrival of the catalogue of products. The ESM allows that the response to the catalogue in the form of a buy list either happens with a delay (for example after the customer selecting via a user interface) or immediately. In the former case, the ESM declares the transition */products*, leading into state *selecting*, from which transition *buy/* leads into state *ordering*.

For the system as specified in Fig. 2, we assume that the order is not delayed, but directly computed by the operations *select products* and *get sCart*. Since these operations are executed locally, they are processed within the same run-to-completion step. This immediate return via *buy* is allowed by the ESM with the transition */products+buy*. Note that *products* and *buy* are streaming parameter

pins denoted in black, meaning that they can pass tokens while the collaboration is active.

The *o:Order* collaboration will eventually finish on the merchant's side via *finish*. This triggers the local operation *create Invoice*, which in turn triggers the start of the payment collaboration. From the ESM for the payment collaboration shown at the bottom right in Fig. 2, we see that this collaboration passes the invoice to the customer, who confirms the payment via *pay*. Since payment information may be invalid, this collaboration may terminate in two different ways, either via *success* or *failed*, upon which corresponding actions are invoked on the merchant and customer sides.

3 The Security-Enhanced Method

The creation of secure systems needs the combined effort of domain experts having in-depth knowledge of the application domain of a system and of security experts [7]. A straightforward way is that the domain-expert first develops a functionally correct, yet unprotected system. Thereafter, the security expert analyzes the system for vulnerabilities, threats and risks, and adds countermeasures hardening it against malicious attacks. This kind of security analysis is well-known since in the seventies [8]. More recently, model-based techniques for secure system development were introduced as well [9–14]. We also propose a technique based on UML [7, 15, 16]. It follows the security analysis standard ISO/IEC Common Criteria [6].

The UML-orientation of the approach makes it easy to be integrated into the SPACE engineering methodology leading to the proceeding depicted in Fig. 3. There the domain expert is expressed by the simple person icon while the security expert is expressed by a person icon carrying a lock.

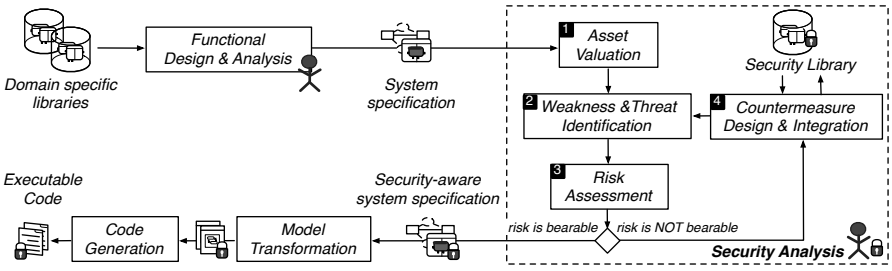

Fig. 3. Security-enhanced development method

First, the domain expert creates a functional system model by composing and analyzing a set of UML collaborations, activities and ESMs, as presented in Sect. 2. Then, when the system is functionally correct, it is handed over to the security expert who enhances the system by performing a security analysis which consists of the following series of steps:

Step 1. Valuation of assets and definition of security objectives.
Step 2. Identification of weaknesses and threats.
Step 3. Assessment of the resulting risks.
Step 4. Planning, design and evaluation of suitable countermeasures.

Step 4 results in an extended system specification which of course may contain new vulnerabilities. Therefore, the analysis has to be reiterated with the extended system model at step 2 which may lead to further countermeasures protecting the original ones. The iteration is stopped in step 3 when all risks for the system are accepted as bearable.

The accomplishment of the steps 2 and 3 can be supported by various tools based on graph transformation [15, 16] as well as model checkers such as Scyther [17] and Casper/FDR [18].The use of these tools, however, is not within the focus of this paper.

The integration of countermeasures in step 4 is supported by a library of basic security primitives which facilitates the development of suitable countermeasures. Moreover, the collaboration models of security mechanisms realizing often used countermeasures can also be stored in a library.[1] Thus, the security expert adds a countermeasure by utilizing the corresponding blocks of the library and composing them with the blocks modeling the system functionality. Sometimes the integration of a security mechanism changes the functionality of the system (see [7]). In that case, the domain expert has to check if the changes can be accepted and the functional analysis has to be repeated for the extended system.

When the security expert decides that the risks of the resulting system are bearable, the system specification is transformed into a component-oriented model from which executable code is generated. Of course, these two transformation steps have to guarantee that no further vulnerabilities are added which, however, is not discussed here.

4 Security Analysis of the e-Sale System

The e-sale system will be implemented as two distributed components. The component for the merchant is run on a server, while the customer component can be deployed on a mobile device or a computer. Therefore, there are three parts that make up the e-sale system: the merchant application, the customer application and a communication channel between them, which we assume to be public.

In the following, we focus on confidentiality and integrity issues in communication security. We assume that sufficiently strong access control (LDAP, X.509) is enforced in the devices running the system components such that direct attacks on the hosts are unlikely. Concerning availability, we assume an IDS is in place protecting the system against denial-of-service attacks.

[1] Examples of building blocks for basic security primitives and security mechanisms are introduced in Sect. 5

In step 1 of the security analysis, the security expert identifies that information exchanged between the components is an important asset. In the UML activity, a message transmission is represented as activity flow crossing a partition border. These flows are hidden in the *Order* and *Payment* services such that the expert has to examine their details in addition to the system in Fig. 2. From the UML activity for the *Order* service, depicted in Fig. 4, three types of information can be distinguished: a simple call message for the catalogue of products, the catalogue itself and the order data including information about the items to be purchased and the delivery address. In the *Payment* service (not shown in detail here), there are two types of messages: invoice data and payment information which includes the credit card number, card holder name, expire date and card security code.

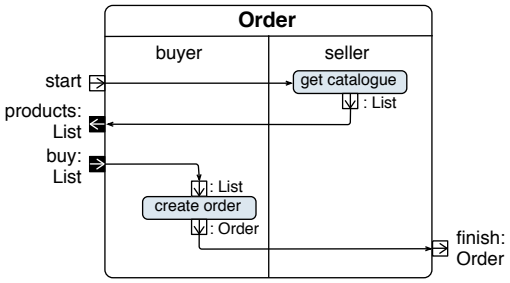

Fig. 4. UML activity for the Order service

In order to estimate the value of an asset, we attach properties related to the basic security objectives *confidentiality* and *integrity* to the asset. The magnitude of these properties describes the financial value of the asset and in correspondence the degree of protection needed. Since it is often difficult to estimate the true financial value of an asset, we use instead seven security levels which correspond to the evaluation assurance levels defined in the Common Criteria [6]. Level 1 should be assigned to the confidentiality property of information if the damage caused by revealing the information is only minor, while level 7 should be used if by eavesdropping leads to highly serious consequences particularly for the owner of the information.

The security expert assigns level 1 to the confidentiality property of the catalogue of products in the e-sale example since it is intended to be publicly available so that everybody can read it easily. The same level also applies for the integrity property. In contrast, the order information has level 5 both for its confidentiality and integrity properties since revealing it to other entities leads to privacy issues and modifying the information also causes serious consequences for both the customer and the merchant. The same holds for the invoice information, while the payment information is rated with level 7 due to the severe consequences of eavesdropping the credit card information.

In step 2 of the security analysis, weaknesses and threats of the system are identified by considering possible attacks. Since the communication channel is public, malicious entities can eavesdrop, alter and replay messages in the channel. All information exchanged is vulnerable since no protection has been applied yet. Thus, those attacks are all substantial threats.

Based on the valuation of an asset and the seriousness of an attack on it, risk is calculated in step 3 of the security analysis. The matrix[2] in Tab. 1 [7, 15] is used to calculate the risk level. It reflects that risks in general depend on both the value of an asset and on the seriousness of the vulnerabilities and threats [19]. For example, since the confidentiality and integrity value of the catalogue of products is 1 and the threat seriousness level is 7, the risk level of this information is 3. For the order information, the result is 6 both for the confidentiality and integrity risks. Using a policy in which level 3 is considered bearable, this risk assessment shows that the risk of transmitting the catalogue can be accepted. However, the risks for the other three assets are too high. Therefore, we proceed to step 4 in which the security expert designs suitable countermeasures to mitigate the threats and integrates them into the system.

Table 1. Matrix for calculating risk values

Security level	Threat seriousness level						
	1	2	3	4	5	6	7
1	0	0	1	1	2	3	3
2	0	1	1	2	3	3	4
3	1	1	2	3	3	4	5
4	1	2	3	3	4	5	5
5	2	3	3	4	5	5	6
6	3	3	4	5	5	6	7
7	3	4	5	5	6	7	7

5 Security Specific Building Blocks

Cryptographic techniques are typically used to protect messages in transit. To achieve confidentiality, one can employ encryption. A digital signature can be used to detect message modifications. Finally, adding a nonce to a message thwarts replay attacks. One can use a combination of these three basic cryptographic primitives to protect the order, invoice and payment messages in the e-sale system. Thus, instead of sending a message msg in clear, the message is encoded in the form: $\{n|msg|\{hash(n|msg)\}_{KR_s}\}_{KU_r}$, where | represents concatenation of data, n is a nonce or a random value, $hash()$ is a one-way hash function, $\{\}_{KR_s}$ represents an encryption with the private key of the sender and

[2] A risk level 0 means that no risk is assumed

$\{\}_{KU_r}$ is an encryption with the public key of the receiver. The rest of this section will show how this solution is represented by SPACE building blocks.

Some basic security primitives consist of some security operations that must be executed by a sender and a receiver to fulfill one or more security objectives. For example, to achieve message integrity, one can use a digital signature that consists of two operations: one for signing the message and the other for verifying the signature. These two operations are performed by the sender and the receiver respectively. We encapsulate this pair of operations in a single building block and put it in a library of basic security primitives in order to help the security expert to create more complex security mechanisms.

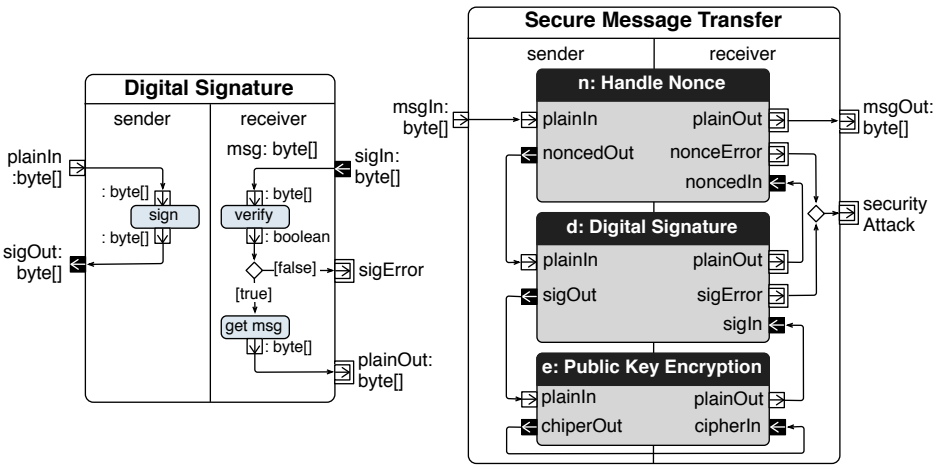

Fig. 5. Building Block for Digital Signature and Secure Message Transfer

The building block for the security primitive digital signature is shown on the left part in Fig. 5. This block starts by receiving a token containing a message m to be signed via pin *plainIn*. Then, a call to an operation *sign* implementing the signing function is performed and a token is emitted via pin *sigOut* with a data in the form of $m|\{hash(m)\}_{KR_s}$. Sometimes later, the receiver receives the signed data via pin *sigIn* and verifies the signature $\{hash(m)\}_{KR_s}$ in the operation *verify*. Further, it stores the original message m in the variable *msg* and outputs a Boolean value indicating whether the integrity of the message is preserved or not. According to the result, either the original message is given out via pin *plainOut* or a token is emitted via pin *sigError*. Note that the *Digital Signature* block does not itself specify any direct communication between its participants, but rather encapsulates corresponding operations.

The operations *sign* and *verify* in the *Digital Signature* block contain Java code for a digital signature implementation with a particular algorithm. We utilize a set of APIs from Java Cryptography Architecture (JCA) [20] and Java

Cryptography Extension (JCE) [21] for the implementation of cryptographic algorithms and other related materials such as encryption keys and public key certificates. It is of course possible to adapt the building block to use other secure implementations of cryptographic algorithms or keys.

Building blocks for *Public Key Encryption* and *Handle Nonce* are also created. They are not shown here in detail, since those blocks are similar to the *Digital Signature*. The security expert then composes instances of these three blocks in order to create the block of security mechanism *Secure Message Transfer*, depicted on the right part in Fig. 5. A message transferred from the sender to the receiver in this block is protected in a way that the transmitted message is unintelligible for entities apart from the sender and the receiver as long as the private key of the receiver is not compromised. This block also guarantees that every message emitted via pin *msgOut* is always identical to the corresponding message received via *msgIn* as long as the private key of the sender is not jeopardized. Furthermore, if a malicious entity attempts to replay a message or make a modification on it, the receiver gets a notification via pin *security Attack*.

Since this security mechanism can be used to secure other systems as well, this block is put in the library of security building blocks for further use. Moreover, the security expert may also use this block to create more complex and specific security mechanisms.

6 Secure e-Sale System

The secure message transfer mechanism is used to protect the exchanged messages containing the order, invoice and payment information in the e-sale system. Therefore, instances of the *Secure Message Transfer* block are composed with the *Order* and *Payment* services in order to produce a security-aware system specification. The *Secure Order* service, depicted on the upper right in Fig. 6, is the result of composing the security block with the *Order* service.Similarly, the *Payment* service is also extended, resulting in the *Secure Payment* service.

Note that the *Secure Message Transfer* block indicates a security breach by emitting a token via pin *securityAttack* instead of giving out the transferred message. This has to be taken care of to keep the *Secure Order* and *Secure Payment* services consistent. For the *Secure Order* service, an alternative output pin *securityAttack* is added.

These change alters the functional behavior of the e-sale system. The operation *handle attack*, which contains logging the attacks for further analysis, and the operation *confirm failed*, in which a notification of transaction failure is sent to the customer, are executed respectively in the event of a security breach as shown on the left part in Fig. 6. These enhancements are examined by the domain expert and functional analysis is performed once more to ensure that the secure e-sale system is functionally correct.

Another round of the security analysis needs to be performed. A reiterated step 2 of the security analysis on the secure e-sale system shows that the same threats are still applicable. The malicious entities may still harm the system by

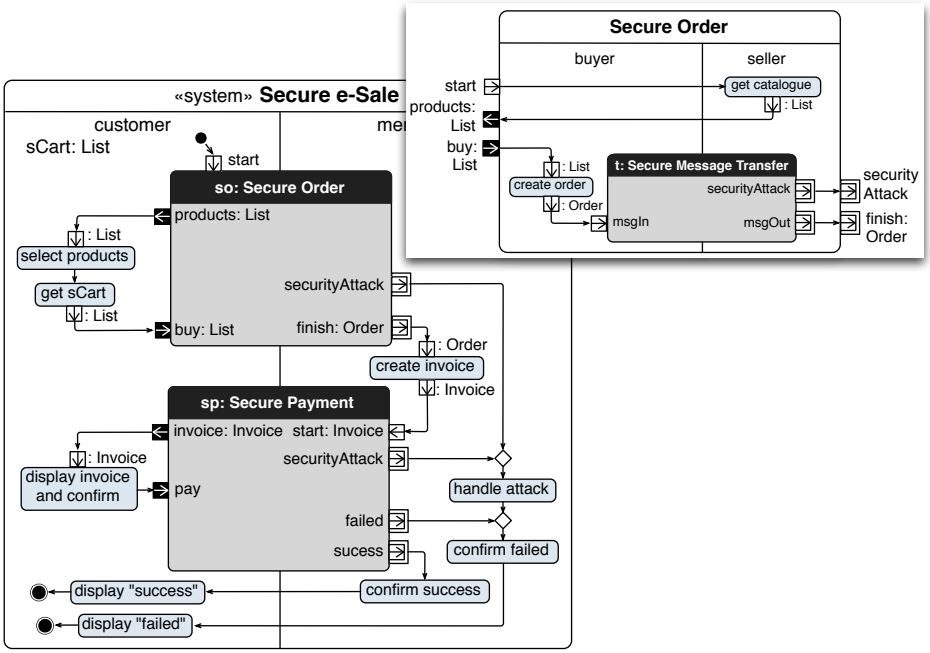

Fig. 6. The secure e-Sale system

eavesdropping, modifying and replaying messages transferred in the communication channel. However, since the security expert employs the secure message transfer to protect the order, invoice and payment information, the threat level is reduced to 1. In consequence, the risk level of those information is now 3 or less and thus bearable. Up to this point, both the domain and security experts are convinced that the secure e-sale system is functionally correct and adequately protected with respect to the risk assessment.[3]

As described previously, the subsequent implementation of the secure e-sale system is performed in two automated steps. First, it is transformed into a component-oriented model, expressed in executable state machines. Afterwards, a platform dependent executable code is generated from the state machines (see [3]). Then, the code including some security settings (e.g., digital certificates) can be deployed.

Collaboration-oriented modeling in our approach facilitates the reuse of security solutions. From our development method, we can classify a security-aware system model into building blocks for security primitives and mechanisms obtained from the security library and thus intended to be reused, application-related blocks and some adaptations needed as the result of security analysis. In order to estimate the degree of reuse related to security aspects we compute

[3] The unprotected transmission of the success respective failed information is also a vulnerability since an intruder can alter it. We do not discuss that here for brevity.

the ratio of the first category to the combination of the last two. Since our development activities consists of creating UML models and writing Java code in the call operation actions as well as other related code (e.g., class Order in the *Secure Order* block), we can estimate the reuse rate in terms of these two factors.

For our secure e-sale system, we use three instances of the security-specific block *Secure Message Transfer*. This block contains the *Digital Signature*, the *Public Key Encryption* and the *Handling Nonce* blocks. The building blocks in the e-Sale system depicted in Fig. 2 are the application-related blocks, while some call operation actions, such as *handle attack*, and pins (e.g., *security Attack*) constitute the adaptations. By calculating the number of UML elements and the number of lines of Java code in the system, we find that 68% of the elements and 32% of the code are reusable security solutions.

Naturally, the degree of reuse depends on many factors. The complexity of security mechanisms is one of them. Moreover, different applications will also result in different reuse proportions. Overall, the reuse of security aspects and also the reuse proportion from functional models (see [5]) contribute to reducing the time and effort in developing security-aware systems.

7 Related Work

Although many experts agree on the importance of integrating security aspects early in the development of a secure system, in practice security is still treated as an add-on since currently there is no silver bullet methodology and tool support to achieve this [22, 23]. Nevertheless, several approaches towards this goal have been proposed.

SecureUML [24, 25] is an extension of UML to specify role-based access control policies. A model transformation can be applied to an extended UML diagram to generate system code that includes security infrastructure. Our work is different from this in that SecureUML is specific for access control.

UMLsec [26] is an extension of UML that models security requirements, such as secrecy secure information flow, secure communication link, etc. as stereotypes, tagged values and constraints. This UML profile can be attached in model elements of UML diagrams. It provides a means to evaluate a UML specification for security. Our approach is different in that UMLsec does not facilitate reuse of a secure system specification. Moreover, it does not specify the automatic generation of code from design models.

In the aspect-oriented modeling, security mechanisms are modeled as aspects and are weaved into base specifications at join points. Mouheb et al. propose a mechanism to weave security aspects into UML 2.0 models at the design phase [27]. Georg et al. propose using the aspect-oriented technology in combination with misuse models in order to perform security analysis [28]. Pavlich-Mariscal et al. propose an approach to extend UML with security diagrams that represent access control policies as aspects [29]. Our work is different in that the aspect-oriented method does not consider the changes of the functional behaviors of the systems after security aspects are weaved into base specifications.

In addition, the CORAS method [30] that contains seven steps of security analysis was proposed. Recently, Refsdal and Stølen suggested an approach that extends a security analysis to include the measurement of related key indicators to determine the likelihood and consequences of unwanted incidents [14]. Differently from those work, our approach also covers the integration of security analysis into system design and implementation.

8 Concluding Remarks and Future Work

We propose an approach for integrating security aspects into system development using collaboration-oriented models. The approach extends the SPACE engineering method by including the asset-oriented security analysis. Particularly, we demonstrate how security analysis is performed on a collaboration-oriented system specification and how security mechanisms are designed as collaboration models and integrated with the system specification.

Our initial experience shows that the collaboration models of security solutions as presented in Sect. 5 are useful for developing security-aware systems efficiently. The building blocks hide the complexity of the security mechanisms facilitating their integration into the system specifications. Moreover, the security library facilitates reuse of some countermeasures and the design of more advanced security solutions.

The approach presented in this paper is not only well-suited for job separation between domain specific experts and security experts, but also provides a mechanism to integrate the work of this two types of experts. Another benefit of this approach is that it allows security solutions to be integrated early in the development of secure systems.

In the future, our approach will be extended in various ways. Of course, we will expand our libraries with building blocks for more complex security mechanisms like authentication, access control, and intrusion detection. Further, it is worthwhile to support carrying out the security analysis steps (see Sect. 3) by meaningful tools. The formal semantics of the SPACE method based on Temporal Logic makes it suitable to model checking. We already integrated a model checker for functional properties to SPACE (see [4]). Likewise, one can attach security-related checkers such as Scyther [17] that inspect the UML models for the existence of vulnerabilities.

In addition, the graphical nature of the model descriptions in the form of UML collaborations and activities leads to the utilization of graph rewriting techniques. For example, in [15] we presented the use of graph rewriting for information flow analysis based on the Decentralized Labeling approach by Myers [31]. There, both information and components are attached with static or dynamic labels (see [32]). Special operators are used to check if information may reach components providing access to principals not allowed to read it. This approach can be added to our proposed method by attaching the labels to the components and to the edges over which the information is passed.

The integration of security mechanisms can also be supported by graph rewriting. For instance, the adding of the building block *Secure Message Transfer* as a call behavior action can be realized by a quite simple graph rewriting rule which, e.g., transforms the model in Fig. 4 automatically to the *Secure Order* service in Fig. 6.

These and further analysis extensions may help to reduce the costs of risk analysis and the integration of suitable security mechanisms to distributed systems. The lower costs may lead to a broader utilization of security analysis and, in consequence, help to rise the security quality standard of software in general.

References

1. Anderson, R.J.: Security Engineering: A Guide to Building Dependable Distributed Systems. John Wiley & Sons, Inc., New York, NY, USA (2008)
2. Herrmann, P., Kraemer, F.A.: Design of Trusted Systems with Reusable Collaboration Models. In: Proceedings of the Joint iTrust and PST Conferences on Privacy, Trust Management and Security (IFIPTM07). IFIP 238, Moncton, New Brunswick, IFIP, Springer-Verlag (July/August 2007) 317–332
3. Kraemer, F.A.: Engineering Reactive Systems: A Compositional and Model-Driven Method Based on Collaborative Building Blocks. PhD thesis, Norwegian University of Science and Technology (August 2008)
4. Kraemer, F.A., Slåtten, V., Herrmann, P.: Tool Support for the Rapid Composition, Analysis and Implementation of Reactive Services. Journal of Systems and Software (2009) (to appear).
5. Kraemer, F.A., Herrmann, P.: Automated Encapsulation of UML Activities for Incremental Development and Verification. In Schürr, A., Selic, B., eds.: Proceedings of the 12th Int. Conference on Model Driven Engineering, Languages and Systems (Models), Denver, Colorado, USA, October 4-9, 2009. Volume 5795 of Lecture Notes in Computer Science., Springer (2009)
6. ISO/IEC: Common Criteria for Information Technology Security Evaluation. (1998) International Standard ISO/IEC 15408.
7. Herrmann, P., Herrmann, G.: Security-Oriented Refinement of Business Processes. Electronic Commerce Research Journal **6**(3–4) (2006) 305–335
8. Baskerville, R.: Information Systems Security Design Methods: Implications for Information Systems Development. ACM Computing Surveys **25**(4) (December 1993) 375–414
9. Baskerville, R.: Designing Information Systems Security. Wiley & Sons, Chichester (1988)
10. CCTA: SSADM-CRAMM Subject Guide for SSADM Version 3 and CRAMM Version 2. CCTA, London. (1991)
11. Kienzle, D.M., Wulf, W.A.: A Practical Approach to Security Assessment. In: Proceedings of the Workshop New Security Paradigms '97, Lake District (1997)
12. Leiwo, J., Gamage, C., Zheng, Y.: Harmonizer — A Tool for Processing Information Security Requirements in Organization. In: Proceedings of the 3rd Nordic Workshop on Secure Computer Systems (NORDSEC'98), Trondheim (1998)
13. Lund, M.S., den Braber, F., Stølen, K.: Maintaining Results from Security Assessments. In: Proceedings of the 7th European Conference on Software Maintenance and Reengineering (CSMR2003), IEEE Computer Society Press (2003) 341–350

14. Refsdal, A., Stølen, K.: Employing key indicators to provide a dynamic risk picture with a notion of confidence. In: Trust Management III, Boston, Springer (2009)
15. Herrmann, P.: Information Flow Analysis of Component-Structured Applications. In: Proceedings of the 17th Annual Computer Security Applications Conference (ACSAC'2001), New Orleans, ACM SIGSAC, IEEE Computer Society Press (December 2001) 45–54
16. Herrmann, P., Krumm, H.: Object-oriented security analysis and modeling. In: Proceedings of the 9th International Conference on Telecommunication Systems — Modelling and Analysis, Dallas, ATSMA, IFIP (March 2001) 21–32
17. http://people.inf.ethz.ch/cremersc/scyther/
18. http://web.comlab.ox.ac.uk/people/gavin.lowe/Security/Casper/
19. Courtney, R.: Security Risk Assessment in Electronic Data Processing. In: AFIPS Conference Proceedings of the National Computer Conference 46, Arlington, AFIPS (1977) 97–104
20. http://java.sun.com/javase/6/docs/technotes/guides/security/crypto/CryptoSpec.html
21. http://java.sun.com/j2se/1.5.0/docs/guide/security/jce/JCERefGuide.html
22. Siponen, M., Heikka, J.: Do secure information system design methods provide adequate modeling support? Information and Software Technology **50**(9-10) (2008)
23. Vaughn, Jr., R.B., Henning, R., Fox, K.: An empirical study of industrial security-engineering practices. Journal of System and Software **61**(3) (2002) 225–232
24. Basin, D., Doser, J., Lodderstedt, T.: Model driven security: From uml models to access control infrastructures. ACM Transactions on Software Engineering Methodology **15**(1) (2006) 39–91
25. Lodderstedt, T., Basin, D.A., Doser, J.: Secureuml: A uml-based modeling language for model-driven security. In: UML '02: Proceedings of the 5th International Conference on The Unified Modeling Language, London, UK, Springer-Verlag (2002) 426–441
26. Jürjens, J.: Secure System Development with UML. Springer-Verlag (2004)
27. Mouheb, D., Talhi, C., Lima, V., Debbabi, M., Wang, L., Pourzandi, M.: Weaving security aspects into uml 2.0 design models. In: AOM '09: Proceedings of the 13th workshop on Aspect-oriented modeling, New York, NY, USA, ACM (2009) 7–12
28. Georg, G., Ray, I., Anastasakis, K., Bordbar, B., Toahchoodee, M., Houmb, S.H.: An aspect-oriented methodology for designing secure applications. Information and Software Technology **51**(5) (2009) 846 – 864 SPECIAL ISSUE: Model-Driven Development for Secure Information Systems.
29. Pavlich-Mariscal, J., Michel, L., Demurjian, S.: Enchancing uml to model custom security aspects. In: AOM '07: Proceedings of the 11th workshop on Aspect-oriented modeling. (2007)
30. Braber, F., Hogganvik, I., Lund, M.S., Stølen, K., Vraalsen, F.: Model-based security analysis in seven steps — a guided tour to the coras method. BT Technology Journal **25**(1) (2007) 101–117
31. Myers, A.C.: JFlow: Practical Mostly-Static Information Flow Control. In: Proceedings of the 26th ACM Symposium on Principles of Programming Languages (POPL'99), San Antonio (1999)
32. Zheng, L., Myers, A.C.: Dynamic security labels and static information flow control. International Journal of Information Security **6**(2) (2007) 67–84