Software Architectural Design meets Security Engineering

Stephan Bode Anja Fischer Winfried Kühnhauser Matthias Riebisch

Technical University of Ilmenau
Ilmenau, Germany
{stephan.bode, a.fischer, winfried.kuehnhauser, matthias.riebisch}@tu-ilmenau.de

Abstract

Security requirements strongly influence the architectural design of complex IT systems in a similar way as other non-functional requirements. Both security engineering as well as software engineering provide methods to deal with such requirements. However, there is still a critical gap concerning the integration of the methods of these separate fields. In this paper we close this gap with respect to security requirements by proposing a method that combines software engineering approaches with state-of-the-art security engineering principles. This method establishes an explicit alignment between the non-functional goal, the principles in the field of security engineering, and the implementation of a security architecture. The method aims at designing a system's security architecture based on a small, precisely defined, and application-specific trusted computing base. We illustrate this method by means of a case study which describes distributed enterprise resource planning systems using web services to implement business processes across company boundaries.

1. Introduction

Software systems have to fulfil requirements with a high complexity. Business systems for example are frequently integrated with other technical systems. Due to the strong interconnection between enterprises and their business processes, systems have to be opened for instance to other parties and to the internet. A car insurance system for example has to cooperate with services for value estimation and with web services for contract management. The enterprise resource planning (ERP) system of a manufacturer that performs the outsourced production has to interact with the ERP systems of its customers. In many cases such software systems play a business critical role. Security threats constitute an immense risk for the business of a company and for its economic perspective.

In order to prevent security risks of the software systems, an early consideration of security issues and an introduction of security technology during the design provide much more efficient, more effective, and quicker results than add-ons to the system's implementation. Nowadays it is common sense that the software and the system architecture have an important influence on the non-functional properties of the system under development. An early consideration of non-functional goals reduces the development effort and time.

Unfortunately, the methods, techniques, and processes of security engineering and the ones of software architectural design are fairly isolated from each other. This leads to the situation that the experts from both fields are mostly not familiar with the basic principles of the other fields. We address this issue by combining software engineering, which considers architectural quality and the realisation of non-functional requirements, with security engineering, which is dealing especially with the security of systems through policies, models, and architecture but is not so well integrated into development processes.

In this paper we introduce a method that integrates software engineering and security engineering. We show the consequences for the development process of software architectures that result from a special consideration of security requirements. We present a methodical way how to get from security goals to the realising solutions in the software architecture. Therefore, the presented method follows the intention to develop security requirements into a security policy, further a security model, and later into a secure software architecture using different mechanisms to fulfil the goals. Along this way, several architectural decisions about concrete realisations have to be made. We will discuss these decisions with regard to alternative solutions and trade-offs between competing non-functional properties. To evaluate our method, we present a case study from an industrial project for building a secure software architecture. The case study deals with an enterprise resource planning (ERP) system, which is using web services.

First we briefly outline relevant state-of-the-art methods for architectural design and for security engineering. In Section 4 we present the method, where we first introduce the case study and the core concepts including the Goal Solution Scheme. The Sections 4.4 to 4.9 represent the phases of the method.

2. Architectural design for non-functional properties

The majority of the software engineering methodologies and processes considers the design of software architectures as an experience-driven process, which has to be specific to every single project. However, recent works in the fields of component-based approaches and software product lines identify common aspects that can be applied to software architectural design in general. Since a functional decomposition is part of the well-known approaches to software design, the functional requirements are usually considered sufficiently. The non-functional requirements, however, tend to lead to complicated solutions with a low architectural quality. Therefore, they require a stronger consideration by the methods.

For developing a software architecture for both kinds of requirements a frequently performed way of construction is described by Bosch's quality attribute-oriented software architecture (QASAR) design method [5]. The method describes three steps. In the first step the functional requirements are implemented by functional components with the Functionality-based Architectural Design (FAD) method. FAD uses core abstractions of functional concepts — the so-called archetypes — to derive architectural components. In the second step, the developed architecture is assessed in order to decide whether the non-functional requirements are fulfilled or not. Different approaches for the assessment of the non-functional requirements are scenario-based evaluation, simulation, mathematical modelling or objective reasoning. Once the non-functional properties of the architecture are assessed, in the third step, the architecture is transformed to satisfy the non-functional requirements specifications. Therefore, suitable functional structures and components are developed for the implementation of as many as possible non-functional requirements. All remaining non-functional requirements are implemented by changing all affected components. In this step the changes are scattered over the system.

We will use the core concept of QASAR — the fulfilling of non-functional requirements by functional solutions — on our approach. However, the steps of QASAR, especially the scattered implementation of the remaining non-functional requirements according to the third step of Bosch's way demand for a very high number of traceability links and hampers maintainability. Moreover, the scattering violates the major principles for secure software systems like the isolation of security-relevant functions. This is one issue we want to address with our approach by carefully considering non-functional requirements.

Another design approach for dealing with non-functional requirements is the Non-Functional Requirements (NFR) framework by Chung et al. [9]. The framework uses so-called softgoals that represent non-functional requirements. Softgoals are goals that have no clear-cut definition or criteria for their satisfaction. This concept considers that criteria for NFR are rather soft and imprecise. The softgoals with their interdependencies are arranged in a Softgoal Interdependency Graph (SIG).

The NFR framework describes several interleaving and iterative activities to build a SIG and based on it a quality software architecture [26, 9]. First NFR softgoals are established and refined into sub-goals by decomposition. Then, different architectural alternatives are developed as so-called operationalisations for the softgoals. In order to be able to choose between different alternatives for the goals in a well-founded way the corresponding design trade-offs and rationale are elaborated. Therefore, correlations that represent conflict or harmony among softgoals as implicit interdependencies are detected. Beyond, softgoals are also refined by argumentations that base on domain characteristics and developers' expertise. Further, the architect has to decide about the criticality of different goals. Finally, adequate solutions are selected and the impact of the decisions is evaluated regarding the solutions' contribution to the non-functional requirements.

In this way the NFR framework with its Softgoal Interdependency Graphs helps to build software architectures that explicitly consider non-functional requirements right from the beginning. However, the NFR framework deals with goals that are soft in their specification, which is critical for security. Therefore, in our approach we use security policies with precisely defined requirements. Further, we consider general design principles in our decomposition and refinement, which Chung et al. do not.

For software architectural design — as to all other engineering disciplines — the reuse of previously developed, standardised solutions is of great importance for various reasons. First of all, their use helps to master the complexity of the design task, i.e. by fulfilling the engineering principle of conceptual integrity. Further advantages arise from the accumulation of knowledge and from the increased efficiency. There are several categories of reusable solutions for software architectural design. The first group covers abstract solutions, which have to be performed, implemented or instantiated to become part of solutions, e.g. solution templates and methods [12]. Examples for software engineering principles are, modularity and separation of concerns, and for architectural patterns and styles, layers and blackboard, and design patterns like observer and factory. The second group of software products and tools comprises components, which are helpful for the architect's work or can contribute to the solution.

In this paper, we classify these solutions according to the non-functional requirements they influence. We extend the architectural design process in a way that these solutions are applied in a systematic way. For the field of security engineering there are principles and abstract solutions, which can be applied in a similar way as mentioned above. We will later introduce the Goal Solution Scheme as a way of classification and assignment of these solutions according to the non-functional goals that are facilitated or hampered by them.

3. Security engineering methods

IT systems with advanced security requirements increasingly apply problem-specific security policies for describing, analysing, and implementing security properties [8, 18, 14, 19, 20, 11]. Security policies are comprehensive sets of rules that are designed to meet a system's security objectives. In order to precisely describe security policies, state-of-the-art IT systems apply formal security models such as [3, 15, 6, 22, 10, 24, 11], allowing for formal analysis of security properties and serving as specifications from which policy implementations are generated [7].

Security policies and their formal models provide quite sharper specifications of a system's security properties than assumed by a SIG. They precisely determine the functional complexity of a system's *trusted computing base* (TCB): those functional components that enforce and protect a system's security policies. Correctness and tamperproofness of a TCB thus are essential prerequisites for achieving a system's security goals and have a commanding influence on the design and implementation of its security architecture.

While security policies and their formal models achieve considerable improvements in the effectiveness, efficiency, and correctness of a system's security properties, there are still major drawbacks that get in the way of a general practical use. From a security engineering point of view, experience with experimental systems shows that because of the large semantical gap between (informal) security requirements and the security mechanisms of today's implementation platforms (operating systems and middleware platforms), the development of security policies, their implementation, and their integration in a TCB are yet highly complex and expensive [20, 23, 11]. From a software architecture point of view, current operating system and middleware architectures that are capable of integrating security policies incorporate large quantities of trusted code, including code written by an arbitrary number of hardware equipment and software vendors. Because a failure in any individual code line of the TCB has the potential of violating the entire TCB integrity, such approaches severely offend the third fundamental reference monitor principle [2, 16]. In consequence, current systems with advanced security requirements still exhibit a large number of security flaws, and validation or verification of their security properties still is a highly complex effort.

In this paper, we propose a security engineering approach that aims at designing a system's software architecture based on a small and precisely identified, application-specific TCB. The approach is based on the idea of replacing current off-the-shelf, large-scale, general-purpose TCBs by tailored, application-specific TCB's with a reduced functionality that is directly derived from the system's security policies.

4. Aligning architectural design with security engineering

In this section we present our method, which aligns architectural design with security engineering. We take up the approaches, concepts, and challenges discussed in Sections 2 and 3 and provide a methodical way that integrates both. We give an overview of the design process and go into the details of several design steps. All described activities of the method are clarified with the help of ongoing examples from a web service case study, which is introduced in the next subsection.

4.1. Web service case study

Our case study describes distributed enterprise resource planning systems providing services to extend business processes of cooperating organisations across company boundaries. In this paper, we especially focus on logistic business processes for cooperative order processing. One important task in this case study is the process of availability checking (*available-to-promise, ATP*), which is a process crossing several company boundaries whenever sub-contractors are involved.

Distributed ATP services provide the examples we use in this paper. In the course of processing a customer's order request, local ERP systems call ATP services for distributed availability checking. The ATP services are implemented as *web services*, receiving customer order requests, forwarding them to corresponding suppliers, analysing the offers, and sending the results back to the ERP system.

In this scenario it is important to note, that not only suppliers may be competing but also customers. Therefore, it is essential that the offers made by the suppliers cannot be read by any customer but the requesting one. Furthermore, it must not be possible that customers or suppliers, except for the offering one, are able to modify any offer.

4.2. Method overview

In this section we give a short overview of the steps of our integrated method. Figure 1 shows an activity diagram illustrating the sequence of design steps for one iteration of the process.

First of all, there must be a *requirements analysis* for a project, wherein the requirements are refined and prioritised. Based on it, the *security policy* with its rules for example for authentication and authorisation is defined. Once the policy is set up, the further steps *security modelling*, *decomposition*, *usage-driven refinement*, and *derivation of security architecture* can be performed. The methodical way finally concludes in the *implementation and validation* of the security properties and architecture. These steps are the important design activities for dealing with the non-functional goal security. In the next sections we discuss requirements analysis and policy development, further we focus on decomposition, usage-driven refinement, and architecture derivation, which do not necessarily require a security model. We do not discuss the modelling step, because it is out of the scope of this paper and there are state-of-the-art works dealing with security modelling [22, 4, 21], which can be integrated in our method. However, the formalised description of policy rules in a model is essential for a later validation.

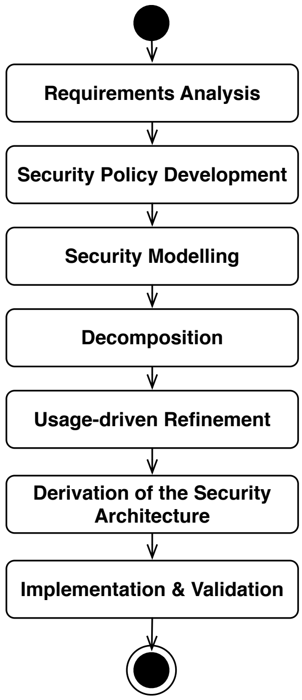

Figure 1. Activities of one iteration of the security architecting process.

4.3. Goal Solution Scheme

As discussed above, in architectural design methods the support for an adequate treatment of non-functional goals like flexibility, scalability or security is rare. Therefore, the authors developed a Goal Solution Scheme, similar to tree diagrams used for example in the Failure Mode and Effect Analysis (FMEA) [25], for a representation of the relationships between different non-functional goals and the development of appropriate solutions by aligning software architectural and security engineering principles.

The Goal Solution Scheme (GSS) covers similar ideas like the Softgoal Interdependency Graph (see Section 2). As shown in Figure 2, the GSS maps non-functional goals to their sub-goals, to principles supporting these goals, and further to functional solutions and technical components for an implementation of the principles. As a consequence, it shows the propagation of non-functional goals, e.g. security, to a software architecture along the design process. From top to bottom the GSS represents possible refinements and decisions during the design and implementation of non-functional goals. The scheme supports different phases in analysis and design and provides guidance for design activities as well as for the establishment and maintenance of traceability links.

The upper transition in Figure 2 supports the detection of conflicting non-functional goals. By a refinement to sub-goals, new interdependencies between the goals such as conflicts become obvious. As a major advantage it facilitates the prioritising for decision-making. The next transition of the scheme guides the designer from non-functional goals to functional or technical principles, thus performing the transformations proposed by the QASAR method. One sub-goal is implemented by ideally one principle. However, a one-to-one mapping is hardly possible. Nevertheless, by mapping only a few principles to one sub-goal, the resulting number of traceability links is significantly lower as with an ad-hoc design. Furthermore, the GSS supports the resolution of conflicting non-functional goals by an identification of potential trade-offs and by prioritising the alternatives. In the lower transition, the principles are linked to functional solutions and existing components implementing them. The crossing arrows in Figure 2 indicate the existence of many interdependencies and trade-offs. As a result, the scheme provides an alignment of principles and functional solutions, it classifies solutions and components according to their impact on non-functional goals, and it provides a stock of reusable solutions to the architect and the designer. The solutions serve as a source of proposals for design alternatives while decision-making. To further illustrate the idea of the Goal Solution Scheme we present a specific example for our case study in Section 4.8.

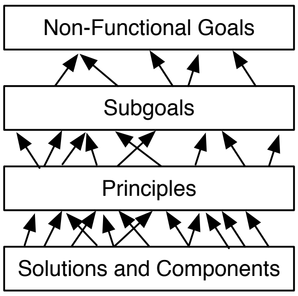

Figure 2. Structure of the Goal Solution Scheme.

4.4. Requirements analysis

Regarding the architectural design process the requirements analysis constitutes an important design step because it provides precisely specified system requirements. Unfortunately, many of the state-of-the-art methods in this field consider non-functional requirements less than functional ones. However, for architectural design we need a balanced consideration of functional and non-functional requirements, including security requirements.

To reduce this deficiency we benefit from security engineering approaches. In this context security properties are specified by security policies and security models. Yet being able to develope a security policy a thorough understanding of a system's security requirements is presumed. In the following we illustrate such security requirements. For brevity's sake, firstly we describe the most obvious non-functional requirements and secondly focus on obvious security requirements by means of the ATP example.

The ATP scenario is characterised by a high degree of flexibility, expandability, and scalability regarding the number and type of customers and suppliers. Changes are not limited to the integration of new customers, represented by their systems, and to the addition of their supply chains to the scenario. Already installed supply chains may change such that suppliers are modified, added to, or removed from the supply chain as well.

In order to enable real-time communication or to even guarantee stability of just-in-time production processes, the ATP scenario must cope with special performance requirements.

Due to realising business processes across company boundaries and companies using services on a shared communication platform, security properties are essential prerequisites for this scenario. When companies call these ATP services, business critical data is not only processed by a shared cooperation platform but also transfered via a public, insecure communication medium. Thus, it is essential firstly to guarantee confidentiality and integrity of data within the borders of the shared cooperation platform and secondly to realise confidentiality and integrity of transfered data.

In our ATP scenario several types of information flow, which necessarily need to be controlled, can occur. Between customers and suppliers maintaining a business relationship business process related information may flow across system bounderies. However, if they do not maintain a business relationship, it is important that there must not flow any business related information at all. Note that it is less important if somebody obtains the knowledge that an information flow has occured between several parties. Following, we illustrate these requirements with an example.

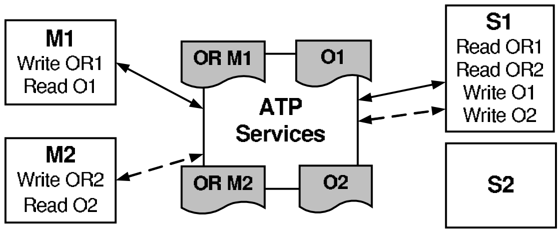

Figure 3. Information flow example within the ATP scenario.

Assuming there are two competing manufacturers of engine starters, called manufacturer M_1 and M_2. Let's further assume they maintain independent business relations to the same supplier S_1 who is connected to the ATP web services and provides both customers with units for their engine starters. If M_1 und M_2 call the ATP services to send a customer order request, S_1 provides an offer O_1 for M_1 and an offer O_2. In order that S_1 is able to make offers, the supplier needs basic data from its customers such as a required number of units, dimensions of units, delivery terms, price limits etc. It is now essential, that this information is only transfered between M_1 and S_1, respectively M_2 and S_1, and cannot be obtained or modified by other customers or suppliers. It is equally important that the offer O_1 is only known to M_1 and O_2 to M_2. Moreover, nobody, except for the offering supplier, must be able to modify the offers O_1 and O_2. On the other hand, it is less important if, for instance, a supplier S_2 knows if S_1 made offers to M_1 and M_2. Figure 3 illustrates the example using continuous arrows to represent legal information flow between M_1 and S_1 and dashed arrows to represent legal information flow between M_2 and S_1.

4.5. Security policy development

Having analysed a system's functional and non-functional requirements, we want to adopt a software engineering method in the next design step. Just as a functional specification is deduced from the system's functional requirements, we deduce a security policy by precisely specifying the security requirements. Following we illustrate the security policy derived from the security requirements of our ATP example.

While a security policy usually includes rules about authentication, authorisation, communication, the handling of persistent data and transaction journals, this paper focusses on its core part, the authorisation policy providing rules to realise information flow control.

In many contemporary systems information flow control is realised by access control applying role-based access control rules. Role-based access control accomplishes confidentiality and integrity while scaling well concerning the number of users, having a friendly, application-oriented abstraction level and many users are familiar with this concept. Therefore, role-based access control suits our requirements very well and information flow control can easily be mapped to role-based access control rules.

User permissions for calling services are derived from service contracts between the distributed ERP system's provider and companies of the users. A user's right to obtain permissions results from his identity; his effective permissions are determined by the roles assigned to him.

To distinguish users of different companies, each user is assigned a *security domain*, a concept adopted from non-interference models [13]. Users of each company are assigned a separate security domain, and users belonging to different companies thus belong to different security domains. Consequently, information flow between different companies is represented by access control rules regarding security domains.

Regarding our ATP scenario this means, users working for manufacturer M_1 belong to a security domain called D_1 and users acting for manufacturer M_2 belong to domain D_2. Thus, a user of D_1 owns the permission to *read* the offer O_1 whereas a user of D_2 only has the read-permission to O_2. By this means, users belonging to domain D_1 cannot access documents of any other domain and vice versa. Figure 3 illustrates the permissions M_1, M_2 and S_1 need to have.

A high level of security may not impose any additional burden on the regular, untrained user. Consequently, the authorisation policy as well as the mechanisms for its enforcement will be integrated transparently in the application. On the other hand, correctness, robustness, tamperproofness, maintainability, and expandability properties of the implementation require that the security policy is rigorously isolated from all other parts of the system. Moreover, the mechanisms for the enforcement of the authorisation policy have to be integrated into the system such that every access to an object is assured. These goals strongly influence the principles which should to be applied. These principles again have a main impact on the overall security architecture.

4.6. Decomposition

One of the most important design activities during the realisation of a secure software system is the step of decomposition. Here the functional decomposition meets the non-functional security requirements. Based on the requirements, the functional decomposition of a software system can be performed with the help of a function tree or a feature model [17], for example by applying Bosch's FAD method that is part of QASAR as stated in Section 2. However, an appropriate balancing of the functional and non-functional issues has to be considered in this design step.

For the functional decomposition of a system that has to fulfil several security requirements, the definition of a security policy as explained in the last section is a necessary prerequisite to guide this step. A security policy as input for the decomposition enables the determination of the functional blocks that are security-relevant and of the ones that are not. To illustrate this categorisation of the functional blocks according to their security relevance, we use a green and red colorisation as symbolisation. To perform the decomposition with special consideration of security requirements right from the beginning, the following algorithmic steps have to be run through:

1. Perform a functional decomposition and mark all functional blocks green.
2. Determine the security-relevant functional blocks and mark them red.
3. In order to further minimise the TCB, red nodes must be decomposed to smaller functional units that can be coloured separately.

As a result of this procedure on the one hand the software system is refined into functional blocks and on the other hand each block is marked as either red or green. Therefore, a categorisation of security-relevant (red) and security-irrelevant blocks (green) is performed. However, in a next step the usage relations between the functional blocks have to be analysed.

For our case study an example could be the following as illustrated in Figure 4, however, grey is used instead of red for printing reasons, and white instead of green correspondingly. If a manufacturer only checks its available suppliers for placing an order request over the ATP service this is not relevant for security. The corresponding feature is marked green. However, as introduced in Section 4.4, where the requirements are analysed, if one places a concrete order request by sending a request to a supplier or wants to read the provided offer this functionality is subject to security constraints. Therefore, the features for this functionality are marked red, as well as the composing ATP service feature.

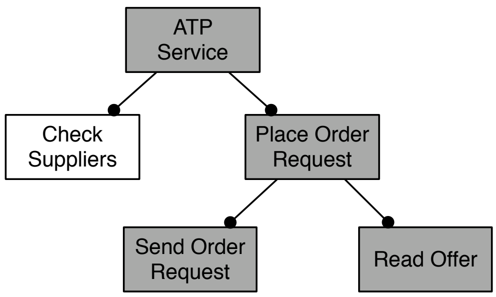

Figure 4. Feature model for the case study.

4.7. Usage-driven refinement

After the decomposition step a graph of functional blocks with a distinction of red and green blocks is available as input for the next. In this design step the usage-relations between the functional blocks are analysed for transitivity. This is necessary because of the transitivity property of the usage relation. The analysis for transitivity is driven by the rules of the security policy. They assure that the security level of the using block cannot be higher as the security level of the block used. As a consequence, usage-relations are only directed from green blocks to red ones — from less security-relevant to equal or more security-relevant functional blocks of the system.

A call for usage directed from a security-relevant block to an irrelevant one results in changing the state of relevance for the called block — in terms of colours a change from green to red. Therefore, in this case, a further refinement of the functional blocks is performed in order to minimise the number of security-relevant blocks, which make up the trusted computing base. With the refinement of a red block some green blocks may be separated and thus the size of the TCB may be reduced.

To continue our case study example, a manufacturer has to select one concrete supplier or even add it when placing an order request. This selection is not security-relevant in contrast to sending the order request to the supplier. Therefore, this function is separated and marked as green to minimise the trusting computing base with the security-relevant functions. The result is shown in Figure 5.

As output from this step of usage-driven refinement we have achieved a clear distinction of the security-irrelevant functional blocks and the security-relevant blocks that belong to the trusted computing base. This serves as the basis for the derivation of the security architecture as the proceeding step.

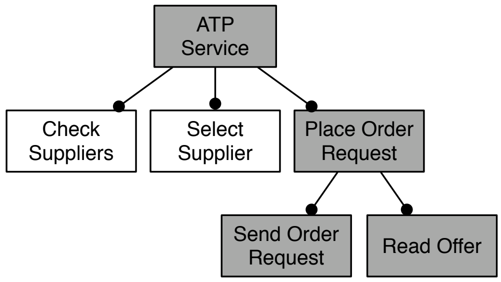

Figure 5. Feature model from Figure 4 after usage-driven refinement.

4.8. Security architecture

For more than two decades, the reference monitor principles [2, 16] have provided fundamental and rigorous guidelines for the design and implementation of security architectures. From a theoretical point of view, a reference monitor simply is an abstract machine that enforces authorised relationships (given by a security policy) between the subjects and objects of an IT system. In practise, reference monitors constitute the inner sanctum of security architectures and generally enforce a system's access control policies.

Three fundamental rules — the *reference monitor principles* — have guided the design of security architectures for many years; these rules require that reference monitors

***RM 1*:** are isolated from any other ordinary system component — the tamperproofness property,

***RM 2*:** totally control any interaction between subjects and objects — the total mediation property,

***RM 3*:** are as small and simple as possible — so that they can be formally analysed.

In an ideal world, these principles would apply to the entire TCB and its implementing security architecture. However, even if any non-security-critical function (a green node in the decomposition graph) is rigorously excluded from a TCB, considering today's excessive number of security mechanisms the rigorous adoption of *RM 1* would result in excessively large security architectures, which then again would violate *RM* 3.

We thus encounter a fundamental conflict between tamperproofness and correctness: by strictly applying *RM* 1, complete encapsulation and isolation of the entire security architecture will result in a high level of tamperproofness of a system's security controls. On the other extreme, any rigorous enforcement of *RM* 3 would result in very small reference monitor implementations, encapsulating only the most critical TCB functions in a well protected part of the security architecture.

As a consequence, the reference monitor principles cannot be applied absolutely; instead the problem has to be approached by criticality-aware security engineering techniques that are well aware of this conflict.

From security requirements to security architectures. Once the security requirements have been casted into a security policy that then guided the colouring of the functional decomposition graph, the sum of all red-coloured subtrees precisely identifies a system's TCB. In the next step, red-coloured subgraphs on the one hand and the reference monitor principles on the other guide the design of the security architecture.

Once again, traditional software engineering meets security engineering. Applying traditional software engineering methods such as architectural styles and design patterns in an unconstrained way would result in software architectures that will fail to exhibit even the most simple qualities of security architectures. Thus we combine both approaches and put the development of the security architecture under the constraints of the reference monitor principles, requiring

- the identification of highly security-sensitive functions to be encapsulated within the reference monitor, pondering tamperproofness against correctness; successful candidates are the red leaves of the decomposition graph (RM 3),
- the encapsulation of the reference monitor into a (hardware- or software-) isolated system component — the inner sanctum (RM 1), and
- the establishment of the total mediation property, providing the reference monitor with the power to control any information flow in the system (RM 2).

As a result, some decisions for a subset of the components are already made. They must not be affected by the following decisions.

Architectural design decisions and traceability. Considering the whole software architecture, decisions on all components and their interfaces have to be made. Those of the red-coloured ones are defined, most of the other interfaces can be derived from information flows, because these interfaces are already pre-determined by them. The components are derived from the functional blocks established in the decomposition and usage-driven refinement steps. Further, the design decisions for software engineering solutions, e.g. architectural styles, design patterns, tools, and standards should be documented. Then, the impact of the decisions for the component architecture, for example decisions about architectural styles or the component model, become traceable.

With the help of traceability links the development steps from the requirements analysis to technical solutions and the implementation can be tracked. The Goal Solution Scheme helps to establish traceability links on the way from the non-functional goals to principles and solutions, and thus helps to document design decisions.

In Figure 6 a Goal Solution Scheme for a case study cutout is shown with traceability links of the type *refine* for the decomposition of the non-functional goals into sub-goals, e.g. security into confidentiality, integrity, and authenticity. Traceability links of the type *realise* are established for the transitions representing a step towards solution of the goals. Therefore, in our case study the role-based access control (RBAC) security policy is linked with the sub-goals confidentiality and integrity it contributes to. Further, the reference monitor principles as well as the principles isolation and minimal TCB that support them are linked, too. The last step shows the assignment of the solutions state machine and access control lists for the RBAC policy as well as a reference monitor for the security principles. Beyond, in the scheme conflicting relations are depicted, for example the trade-off between the first and third reference monitor principle as explained above.

4.9. Evaluation of the architecture

The architectural design has to be evaluated regarding the goals covering both functional and non-functional properties. Regarding security, we have to consider the degree of minimisation of the TCB, its isolation, as well as the total mediation property — the three reference monitor principles.

The established architectural assessment approaches provide appropriate methods and techniques. As one group the questioning techniques include scenario-based evaluations e.g. ATAM, SAAM. The scenarios have to be established in a way that covers an assessment of the reference monitor principles. These techniques fit to an iterative design process because they can be applied in the early design steps, too. As another group, measuring techniques cover metrics as well as simulations, prototypes, and experiments [1]. They can be applied to formal models, e.g. to the information flow model.

The decision which of the techniques is appropriate is determined depending on the progress of the design process and, thus, on the maturity of the architectural artefacts

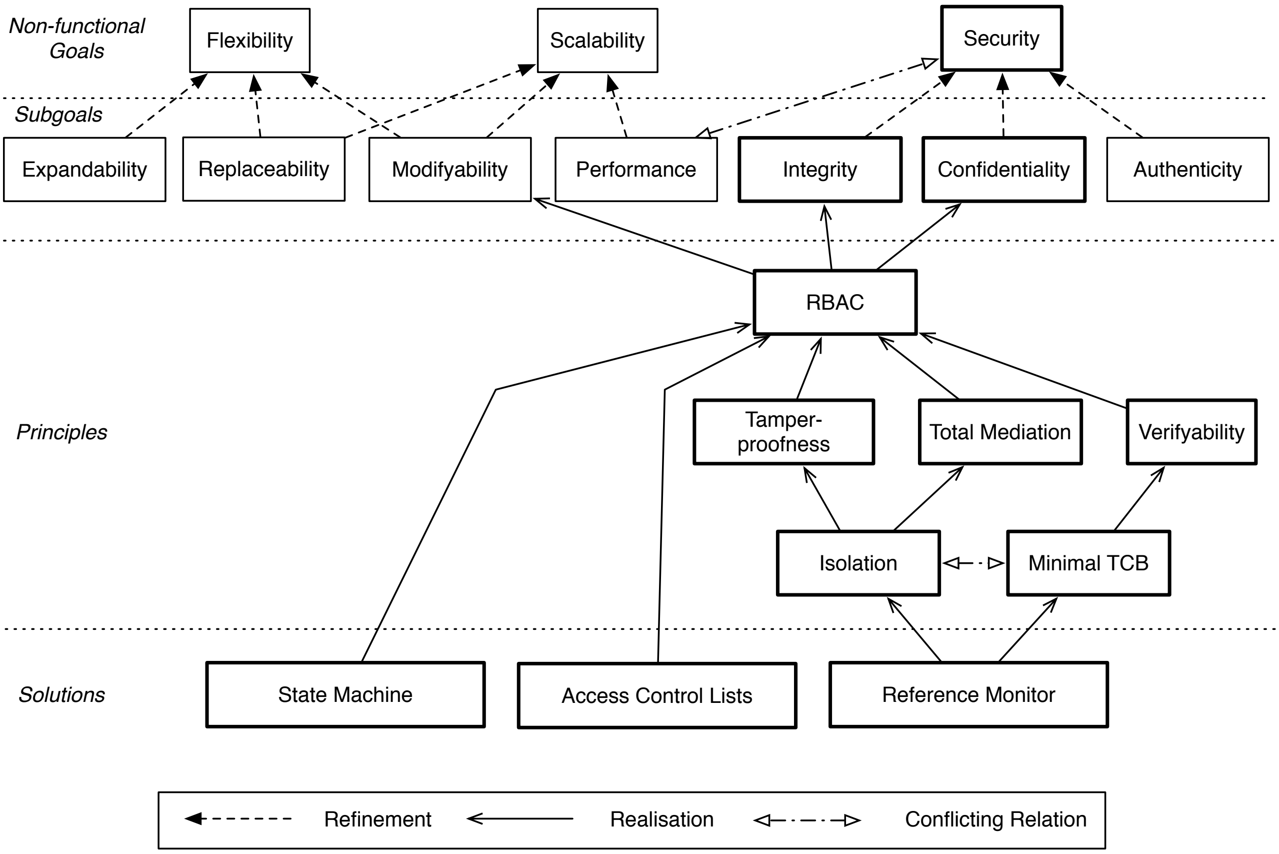

Figure 6. Goal Solution Scheme for the case study.

in terms of completeness and rigour. For example, a semi-formal model of an architecture can only be evaluated by questioning techniques, whereas for a precise model a behavioural analysis can be performed.

5. Conclusion

This paper deals with the architectural design of security-related software systems. We presented a method that guides the developer on a systematic way for the application of security principles and solutions. In this way, an integration of security engineering methods with architectural design methods is achieved. The method supports the analysis and the refinement of the non-functional goals and requirements regarding security, while competing goals are resolved. Both structural and security-related goals are considered, and they are balanced against each other. Established methods and principles of security engineering are applied for fulfilling these goals, e.g. the construction and the explicit reduction of a TCB. The degree of minimisation of the resulting TCB however depends on the application and the amount and the complexity of the security requirements. A Goal Solution Scheme is introduced to visualise the relations between goals, refined goals, and solutions.

As a result, the achieved ease of comprehension for the developer and the clear decision criteria lead to an improved design process. A strict separation between security-relevant and non-relevant blocks in the solution is achieved. The design decisions and their prospective consequences are made clear and explicit. For the latter, three factors are important. Firstly, the indication of the interactions between functional decomposition and security relevance helps the developer during the creative design process. Secondly, the Goal Solution Scheme guides the decisions in a goal-oriented way, especially in the case of competing requirements and trade-offs. Thirdly, the traceability links support the comprehension and facilitate analyses and coverage assessments.

For illustration and proof of feasibility of the method case examples from a web service project in the enterprise resource planning domain are employed.

6. Future Work

Even if design tasks are considered to be creative ones, there is a need for tool support for managing the complexity, for avoiding mistakes, and for effort reduction. The visualisation of the consequences of design decisions during the decomposition step is of special importance, since the functional structuring and the security relevance have a mutual influence on each other. As another aspect of future work we consider a tool support for the exploitation of the security specification, both for implementation and for assessment. During design, tool support is helpful for the management of the traceability links according to the Goal Solution Scheme: the traceability links have to be updated and checked especially during changes and design evolution. For assuring a high design quality in terms of security engineering principles and architectural quality, further works regarding assessments and analyses are necessary. Since the method leads to well-founded decisions, their appropriate implementation has to be assessed. As an example, metrics for an evaluation of a proper decoupling between components and of the degree of minimisation of the TCB have to be mentioned.

References

[1] G. Abowd, L. Bass, P. Clements, R. Kazman, L. Northrop, and A. Zaremski. Recommended Best Industrial Practice for Software Architecture Evaluation. Technical Report CMU/SEI-96-TR-025, CMU/SEI, 1997.

[2] J. P. Anderson. Computer Security Technology Planning Study. Technical Report ESD-TR-73-51, Air Force Electronic Systems Division, Hanscom AFB, Bedford, MA, USA, 1972. Also available as Vol. I, DITCAD-758206. Vol. II DITCAD-772806.

[3] D. E. Bell and L. J. LaPadula. Secure Computer Systems: Mathematical Foundations (Vol.I). Technical Report AD 770 768, MITRE, Bedford, Massachusetts, Nov. 1973.

[4] R. Bhatti, E. Bertino, and A. Ghafoor. A Trust-Based Context-Aware Access Control Model for Web-Services. *Distrib. Parallel Databases*, 18(1):83–105, 2005.

[5] J. Bosch. *Design and Use of Software Architectures: Adopting and Evolving a Product-line Approach*. ACM Press/Addison-Wesley, New York, NY, USA, 2000.

[6] D. F. Brewer and M. J. Nash. The Chinese Wall Security Policy. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 206–214. IEEE Computer Society Press, May 1989.

[7] C. Bryce. The Skippy Security Engineering Framework. Technical Report 1060, GMD – Forschungszentrum Informationstechnik, Mar. 1997.

[8] C. Bryce, W. E. Kühnhauser, R. Amouroux, and M. Lopéz. CWASAR: A European Infrastructure for Secure Electronic Commerce. *Journal of Computer Security, IOS Press*, 5(3):225–235, 1997.

[9] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos. *Non-functional Requirements in Software Engineering*. Kluwer Academic Publishers, 2000.

[10] F. Cuppens and C. Saurel. Specifying a Security Policy: A Case Study. In *Proceedings of the Computer Security Foundations Workshop*, Kenmare, Ireland, 1996. IEEE Press.

[11] P. Efstathopoulos and E. Kohler. Manageable Fine-Grained Information Flow. In *Proc. of the 2008 EuroSys Conference*, pages 301–313. ACM SIGOPS, Apr. 2008.

[12] S. Giesecke, W. Hasselbring, and M. Riebisch. Classifying Architectural Constraints as a Basis for Software Quality Assessment. *Advanced Engineering Informatics*, 21(2):169–179, 2007.

[13] J. Goguen and J. Meseguer. Security Policies and Security Models. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 11–20. IEEE, Apr. 1982.

[14] U. Halfmann and W. E. Kühnhauser. Embedding Security Policies into a Distributed Computing Environment. *Operating Systems Review*, 33(2):51–64, Apr. 1999.

[15] M. A. Harrison, W. L. Ruzzo, and J. D. Ullman. Protection in Operating Systems. *Commun. ACM*, 19(8):461–471, Aug. 1976.

[16] C. E. Irvine. The Reference Monitor Concept as a Unifying Principle in Computer Security Education. In *Proceedings of the IFIP TC11 WG 11.8 First World Conference on Information Security Education*, pages 27–37, 1999.

[17] K. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-021, SEI Institute, Carnegie Mellon University, USA, 1990.

[18] W. E. Kühnhauser. A Classification of Interdomain Actions. *Operating Systems Review*, 32(4):47–61, Oct. 1998.

[19] P. A. Loscocco and S. D. Smalley. Integrating Flexible Support for Security Policies into the Linux Operating System. In C. Cole, editor, *Proceedings of the FREENIX Track, 2001 USENIX Annual Technical Conference, June 25-30, 2001, Boston, Massachusetts, USA*, pages 29–42. USENIX, 2001.

[20] P. A. Loscocco and S. D. Smalley. Meeting Critical Security Objectives with Security-Enhanced Linux. In *Proceedings of the 2001 Ottawa Linux Symposium*, 2001.

[21] Q. Ni, A. Trombetta, E. Bertino, and J. Lobo. Privacy-aware Role Based Access Control. In *SACMAT '07: Proceedings of the 12th ACM symposium on Access control models and technologies*, pages 41–50, New York, NY, USA, 2007. ACM.

[22] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-Based Access Control Models. *IEEE Computer*, 29(2):38–47, 1996.

[23] S. D. Smalley. Configuring the SELinux Policy. Technical Report 02-007, NAI Labs, Jan. 2003.

[24] S. D. Smalley and T. Fraser. A Security Policy Configuration for the Security-Enhanced Linux. Technical report, NAI Labs, Feb. 2001.

[25] D. H. Stamatis. *Failure Mode and Effect Analysis: FMEA from Theory to Execution*. ASQ Quality Press, 2nd edition, 2003.

[26] N. Subramanian and L. Chung. Process-Oriented Metrics for Software Architecture Evolvability. In *Proceedings Sixth International Workshop on Principles of Software Evolution*, pages 65–70, Sept. 2003.