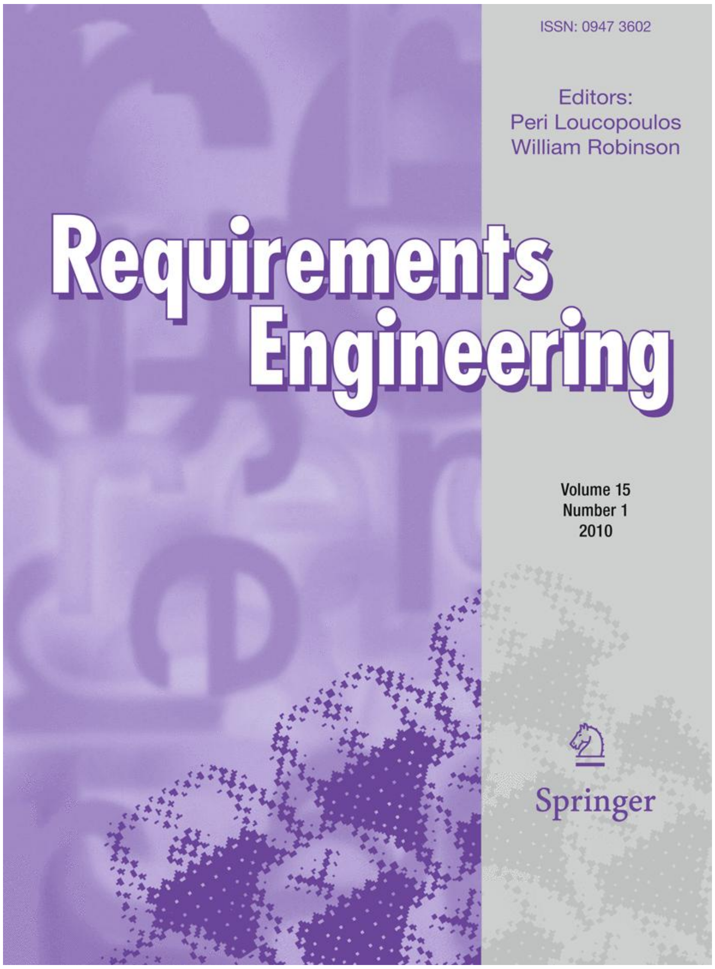
ISSN 0947-3602, Volume 15, Number 1



ISSN: 0947 3602

Editors:
Peri Loucopoulos
William Robinson

Requirements Engineering

Volume 15
Number 1
2010

Springer

This article was published in the above mentioned Springer issue.
The material, including all portions thereof, is protected by copyright;
all rights are held exclusively by Springer Science + Business Media.
The material is for personal use only;
commercial use is not permitted.
Unauthorized reproduction, transfer and/or use
may be a violation of criminal as well as civil law.

Requirements Eng (2010) 15:7–40
DOI 10.1007/s00766-009-0092-x

Author's personal copy

SPECIAL ISSUE - SECURITY REQUIREMENTS ENGINEERING

A comparison of security requirements engineering methods

Benjamin Fabian · Seda Gürses · Maritta Heisel ·
Thomas Santen · Holger Schmidt

Received: 30 October 2008 / Accepted: 4 November 2009 / Published online: 26 November 2009
© Springer-Verlag London Limited 2009

Abstract This paper presents a conceptual framework for security engineering, with a strong focus on security requirements elicitation and analysis. This conceptual framework establishes a clear-cut vocabulary and makes explicit the interrelations between the different concepts and notions used in security engineering. Further, we apply our conceptual framework to compare and evaluate current security requirements engineering approaches, such as the Common Criteria, Secure Tropos, SREP, MSRA, as well as methods based on UML and problem frames. We review these methods and assess them according to different criteria, such as the general approach and scope of the method, its validation, and quality assurance capabilities. Finally, we discuss how these methods are related to the conceptual framework and to one another.

B. Fabian
Institute of Information Systems, Humboldt-Universität zu Berlin, Berlin, Germany
e-mail: bfabian@wiwi.hu-berlin.de

S. Gürses
ESAT/COSIC, K.U. Leuven, Leuven-Heverlee, Belgium
e-mail: seda@esat.kuleuven.be

M. Heisel · H. Schmidt (✉)
Software Engineering, University of Duisburg-Essen, Duisburg, Germany
e-mail: holger.schmidt@uni-duisburg-essen.de

M. Heisel
e-mail: maritta.heisel@uni-duisburg-essen.de

T. Santen
European Microsoft Innovation Center, Aachen, Germany
e-mail: thomas.santen@microsoft.com

Keywords Security requirement · Security requirement engineering · Comparison · Framework for security requirement engineering

1 Introduction

The long-standing credo of requirements engineering reads: “If you don’t know what you want, it’s hard to do it right.” This statement has particular significance for security requirements, because unless we know what to secure, against whom, and to what extent, it is obviously very hard to construct a secure system or to make a substantial statement about its security.

Today, the established way for describing security requirements, as reflected for example in the Common Criteria [1], an international standard to achieve comparability of independent IT security evaluations, starts with a description of the functional requirements, the system architecture, and its working environment. It then continues with a threat analysis that describes envisaged threats, possibly followed by an evaluation of the severity of threats through a risk analysis and ends with the definition of a security policy.

This view on security requirements gives rise to the conjecture that a proper security requirements elicitation is *not* part of the best practice, today. We present two further observations supporting this judgment. The first observation concerns the current process of establishing a security policy, which is supposed to document security requirements. The security policy is derived from a threat analysis whose subject necessarily is a structural description—an architecture or a design—of the technical system to be built. Without information about the intended technical solution, which will implement the functional requirements, there is no handle for a threat analysis to identify possible targets of an attack. Thus, the security policy logically relies on the design of the system.

The second observation concerns the content and role of a security policy, which apparently has flavors of requirements *and* design to it:

- "A security policy is a statement of what is, and what is not, allowed" [2, p. 9];
- "for us, security boils down to enforcing a policy that describes rules for accessing resources" [3, p. 14];
- "the security policy of a system or an organizational unit fixes the set of technical and organizational rules, rules of conduct, responsibilities, and roles, as well as measures to achieve the desired protection goals" [4, Def. 1.14, in German]; but also:
- "based on the risk analysis, the security requirements of the system to be built need to be derived and the security policy needs to be fixed" [4, p. 205, in German].
- "security policy is a [...] policy that mandates system-specific [...] criteria for security [...]" [5, p. 34]

In the terminology of the Common Criteria, a security policy refers to organizational requirements restricting the environment of the technical system. The security requirements are documented in the *security objectives*, which "counter the identified threats and address identified organisational security policies and assumptions" [1, Part 1, p. 29], and the *IT security requirements*, which "are the refinement of the security objectives into a set of security requirements for the TOE [Target of Evaluation] and security requirements for the environment which, if met, will ensure that the TOE can meet its security objectives" [1, Part 1, p. 29].

In those views, security requirements are consequences of threats to the system, which can only be derived from the design of the system. But what makes a threat a threat? There must be an adversary, i.e. someone or something who threatens, and something the threat is directed at—the asset, which is a piece of information or a resource. There also must be someone who values that information or resource and wants it to be protected: the security stakeholder. But most importantly, the security goal that the stakeholder has with respect to the asset must be described in detail. Just to state that the confidentiality, integrity, or availability of the information or resource must be protected is as useful a "requirement" as the often cited prototype of a would-be non-functional requirement: "The system shall have a simple and comprehensible architecture; its usage must be intuitive" [6, p. 274, in German]. Those "requirements" are not verifiable. It is impossible to set up criteria under which a system meets those requirements, because they lack information and are imprecise.

Furthermore, the Common Criteria, and most other suggestions for a security requirements process, identify the stakeholder with the owner of the asset and assign the responsibility to protect the asset to its owner. As a result, the owner of the asset must also be the owner of the IT system. How could he or she otherwise assume responsibility for protecting the asset?

But nowadays, the world is not as simple as that: in civil systems, in which we are interested, there are many more stakeholders who have an interest in an asset than just the owner of the IT system. More often than not, stakeholders have conflicting interests with respect to assets. The paradigm of *multilateral security* [7] acknowledges this fact. Multilateral security contradicts the traditional view, which assumes that there is a "trusted tribe" who has a homogeneous set of security requirements against the rest of the world. But this traditional assumption still heavily influences common approaches toward security engineering.

To take multilateral security seriously in security requirements engineering (SRE), a requirements engineering process must support engineers in identifying security goals of the security stakeholders, and in resolving conflicts among them—and in the reconciliation of security goals and other, notably functional, requirements. This process must establish a coherent set of security requirements for the entire system, which is complete and consistent within itself and with the other kinds of requirements that are relevant for the system. All this must be done *before* the design of the system is fixed, because the security requirements have an influence on the functional requirements, which in turn (hopefully) determine the design of the system. Only after all kinds of requirements have been fixed, threats against assets can be identified, and countermeasures be designed.

Having thus motivated the need for an explicit requirements engineering process for security, this paper contains two contributions. First, we establish a conceptual framework for security requirements engineering in Sect. 2, followed by Sect. 3 on related work. This conceptual framework contains all notions we deem relevant for SRE, as well as their interrelationships. Second, in Sects. 4–9, we give descriptions of currently available methods for SRE and relate these methods to our conceptual framework. However, not only currently available methods can be assessed and compared using our framework but also helps to classify newly developed approaches to SRE. Already today, there is a wide variety of methods covering different aspects of SRE (see Sect. 10). However, important aspects of SRE deserve further research, especially those concerning a multilateral view and conflict resolution. These issues are discussed in the final section.

Table 1 Distribution of SRE papers to years

1999	2000	2001	2002	2003	2004	2005	2006	2007	2008
1	1	5	6	9	13	14	28	11	6

2 Conceptual framework

This section introduces a conceptual framework for security requirements engineering. This conceptual framework (CF) is not an attempt to suggest a universal method that composes existing SRE approaches. In contrast, it describes the central concepts of SRE and their relationships.

The CF should serve as a guideline for comparing different SRE methods, and is itself the result of an iterative process. It started out from reconciling Zave and Jackson's influential terminology [8] with basic concepts from security engineering. Then, we conducted a broader survey on SRE methods (see Sect. 3 for a description of the literature survey), and refined the CF iteratively to cover new concepts encountered during this process. The SRE methods were analyzed again for the survey in Sects. 4–9, based on the final version of the CF.

The utility of the framework lies in the uniform basis it provides to elaborate the specifics of the SRE methods that we investigate in the sections to follow. In particular, mapping the diverse nomenclatures of different methods to the concepts that the framework describes eases the comparison of the methods. Similar surveys of security requirements engineering concepts do exist, which we discuss in Sect. 3.

The scope of the survey was developed following a literature review for which we queried google scholar[1], IEEE Explore[2] and Citeseer[3]. We also consulted the programs of the International Requirements Engineering Conference.[4] After duplicates—i.e., articles with the same title and similar content in different outlets—were removed, we obtained 94 publications that matched our "security requirement" queries and in fact were related to methods for SRE.

The distribution of the papers according to years are given in Table 1. In order to limit the scope of the survey, we included only those articles which had a fully developed method for security requirements engineering. We put the emphasis on software engineering-oriented approaches and did not consider articles that were solely about risk management or security engineering. Last, we picked those methods which were described and validated in multiple papers. Hence, we did not consider one-shot articles on security requirements engineering methods.

Figures 1 and 2 illustrate the relationships between the foundational concepts of security requirements engineering. The concepts are denoted by boxes, which are related by different kinds of lines. Larger boxes make up concept groups. A simple line denotes a relationship between concepts that belong to the same group. A solid arrow denotes a logical dependency between concepts or concept groups, pointing from an antecedent to a consequent. Often, this dependency is a concretization, leading from a more abstract to a more concrete concept. Note that the term *concretization* constitutes a generalization of the term *operationalization*. We use concretization to describe a refinement step, i.e., a concept becomes more detailed; a similar, but restricted sense is sometimes used in requirements engineering for the term operationalization, e.g., turning goals into an operation model (KAOS, GBRAM), which can be verified through testing or formal verification.

In contrast, there is also another established meaning of operationalization (cf. [9–11]), i.e., to express the transformation of non-functional requirements into functional ones. Typically, in both cases, the operationalization of requirements generates a specification (see Sect. 2.4) or the initial system design elements. The term concretization in our sense can be applied to every refinement step.

The solid arrows are labeled when they refer to additional activities in the given dependency such as reconciliation or validation of the requirements. The dashed arrow in Fig. 1 concerns the fulfillment of system requirements by the conjunction of specification, assumptions, and facts, which is discussed in Sect. 2.4 below.

In general, none of the arrows should imply a temporal order of requirements engineering activities. It is understood that all activities to elaborate the different aspects of security requirements take place throughout the entire development process, though with differing intensities.

In the following, we describe the conceptual framework depicted in Figs. 1 and 2 in detail.

2.1 A system is a machine in its environment

Using Zave's and Jackson's terminology [8], a *system* consists of a *machine* in its *environment*. The machine is the technical IT system that is to be constructed and that communicates with its environment. Adopting a holistic view, we consider security to be a system property. Security can only be regarded as a characteristic of a system. It is not a characteristic of the machine alone.

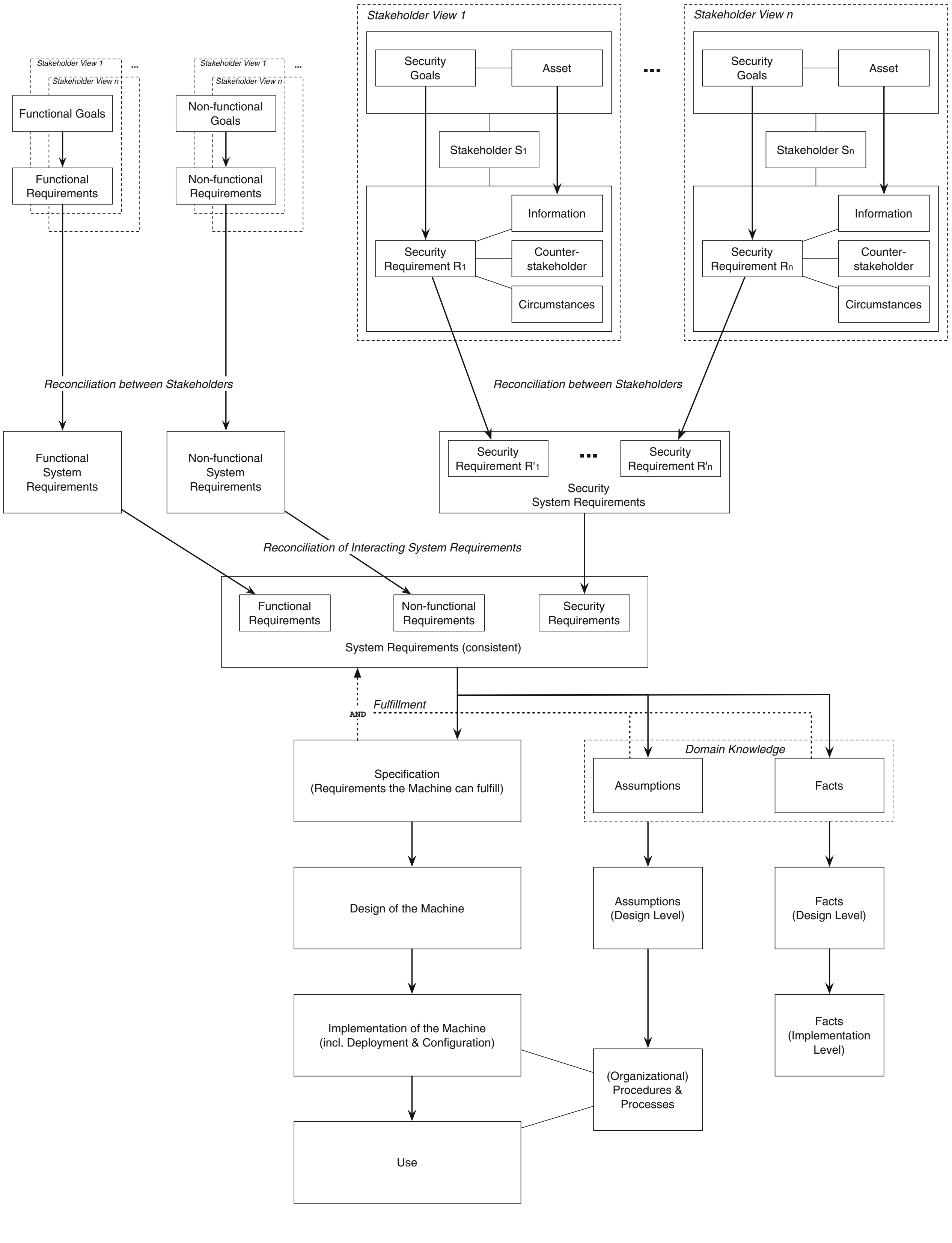
[1] http://www.scholar.google.com

[2] http://www.ieeexplore.ieee.org

[3] http://www.citeseer.ist.psu.edu

[4] http://www.re09.org

Author's personal copy



Note: An arrow primarily denotes a **concretization** and **logical dependancy** between concepts, **not necessarily a temporal succession** of actual process steps.

Fig. 1 Conceptual framework for SRE

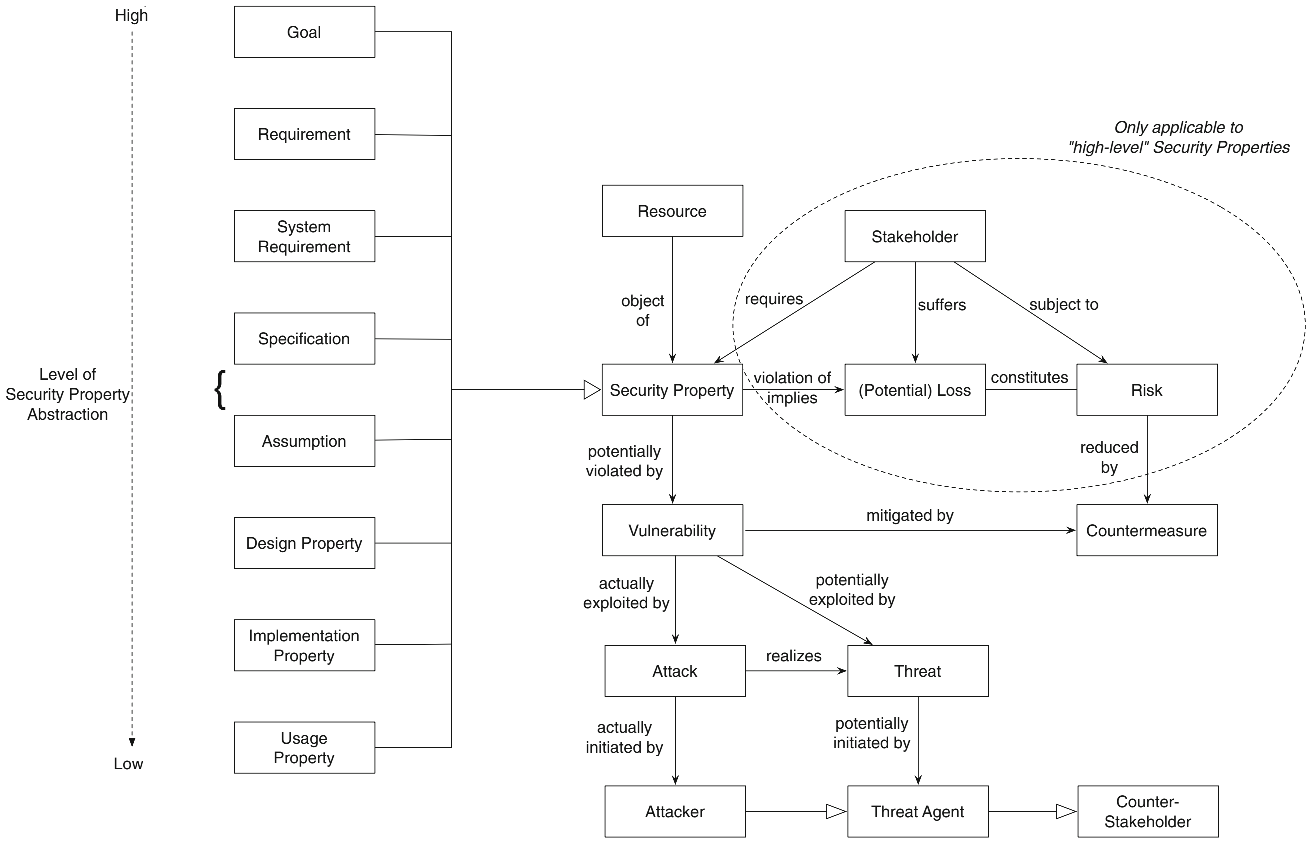


Fig. 2 Security concepts for SRE

2.2 Stakeholder views

A *stakeholder* is an individual, a group, or an organization that has an interest in the system under construction. A *stakeholder view* describes the requirements of a particular stakeholder. The stakeholders may express different *types* of requirements.

For the purpose of our paper, we assume an intuitive distinction between functional ("what the system does" [12, p. 483]) and non-functional requirements, "global requirements on its development or operational costs, performance, reliability, maintainability, portability, robustness and the like" [12 p. 483]. This intuitive dichotomy is also part of established (rather practical) guidelines on the subject [13, p. 119]. However, we have to acknowledge that this distinction is heavily debated in the requirements engineering community because of terminological and conceptual difficulties (see [14] for a comparison of different interpretations), and new categories have been proposed, see e.g. [14, 15]. We consider security requirements to be a part of the non-functional-requirements.

The top row of Fig. 1 distinguishes security stakeholder views from views concerned with functional requirements or non-functional requirements other than security requirements.

Stakeholders can express security concerns at different levels of detail. Therefore, we distinguish security goals (abstract) from security requirements (more detailed)—with the caveat that such a distinction is not readily established in the requirements engineering community, and probably cannot be made completely precise due to vagueness of subjective intuitions and semantic intricacies of natural languages. We give a working description of the differences between security goals and security requirements in the following.

A stakeholder's *security goal* expresses his or her security concerns towards an asset. The Common Criteria [1] define an *asset* as an "entity that the owner of the target of evaluation places value upon". Similarly, ISO/IEC FDIS 17799:2005 [16] and ISO/IEC 13335-1:2004 [17] consider "anything that has value to the organization" an asset. Thus, an asset is any entity that a stakeholder puts a value upon with respect to security. We prefer this definition of an asset over more technical ones that emphasize the value of a technical system to its owner, such as the definition used in NIST SP 800-26 [18], which considers a "major application, general support system, high impact program, physical plant, mission critical system, or a logically related group of systems" an asset. During the security requirements engineering process, assets will be concretized by more detailed concepts, which we subsume under the term *resource*, see Sect. 2.5.

Security goals are traditionally classified into *integrity*, *confidentiality*, and *availability* goals. Very influential examples are the following ISO/IEC 13335-1:2004 definitions:

- *Integrity* is the property of safeguarding the accuracy and completeness of assets.
- *Confidentiality* is the property that information is not made available or disclosed to unauthorized individuals, entities, or processes.
- *Availability* is the property of being accessible and usable upon demand by an authorized entity.

This so-called CIA triad is sometimes extended by concepts such as accountability, non-repudiation, and authentication. In this paper, however, we subsume these further concepts under those of the CIA triad to streamline our presentation. For example, accountability and non-repudiation can be classified as integrity goals, authentication as a design mechanism to achieve confidentiality or integrity, and anonymity or unobservability may be subsumed under confidentiality goals. We emphasize that our conceptual framework could also be applied to different taxonomies and hierarchies of security goals.

Independently of their taxonomy, security goals are defined as very general statements about the security of an asset. For example, the customers of a bank may have the confidentiality goal that their financial situation remains confidential. The public, represented by a government agency, may have the integrity goal that electronic financial transactions do not change the total amount of circulating money. Security goals are therefore too vague to be considered verifiable requirements.

Security requirements capture security goals in more detail. A security requirement refines one or more security goals. It refers to a particular piece of *information* or service that explicates the meaning of the asset it concretizes in the context of the system under construction. This information itself does in general not directly correspond to data that is processed by the machine, e.g., the entries in a database. In turn, it describes a more abstract concept, which the more detailed data will concretize. Concretization of goals to requirements to specifications is accomplished in parallel to the concretization of resources, for example, from assets to information to data.

A security requirement also indicates the *counter-stakeholder* against whom the requirement is directed. This is particularly important for confidentiality requirements, where the counter-stakeholder is the party who must not get to know the information to which the requirement refers. A counter-stakeholder is not necessarily an adversary who tries to attack the system. The concept of an adversary only becomes relevant in the context of a threat analysis, which we discuss in Sect. 2.5.

A third important aspect of a security requirement concerns the *circumstances* in which it must be satisfied. These describe application conditions of functionality, temporal, or spatial aspects, the social relationships between stakeholders—in general, the “context” to which the requirement refers.[5]

In the banking example, a customer’s security requirement states that the exact balance of his or her account must not become known to arbitrary bank employees or other customers. Here, the information is the account balance, and the counter-stakeholders are bank employees and other customers. The circumstances describe, e.g., that an authorized bank employee who needs to know the balance for a specific purpose may nevertheless get to know it, or that the balance must be kept confidential for at least 50 years after the account has been closed.

2.3 System requirements

Multiple requirements are interdependent and interact with one another. These interactions may be positive, negative (conflicts), or result from conflicts during implementation [20]. Analysis and management of requirements interactions is a necessary and fruitful part of requirements engineering [20]. Although analysis of positive interactions may be useful in prioritizing requirements or reiterations in requirements analysis, focus in requirements interaction has been on negative interaction, or simply requirements conflicts.

Conflicts may arise for a number of reasons at different stages of requirements engineering. The views of different stakeholders are in general inconsistent, e.g., because they have different and contradicting requirements. Inconsistent requirements are the starting point for deriving useful information that might otherwise go unnoticed, cf. [21] (as cited by [20]). Inconsistencies resulting from differing views have been addressed by view-point-oriented requirements engineering methods [22, 23]. Conflict of interest, the case in which an individual’s personal interest conflicts with the requirements assigned to their roles is addressed by Giorgini et al. [24].

[5] We propose to avoid the use of “context”, because it is an overloaded term within software engineering. Often in requirements engineering context is used to refer to the specific environment in which the machine is situated [19]. Context has also increasingly become a concept in newer branches of computer science. One example being “context-aware systems” modeling the properties of a given context, which may or may not be the physical environment. Context is then used to adapt the machine or the environment to the users’ needs.

Even when the differences between stakeholders and the roles that are assigned to them are consolidated, the satisfaction of one requirement can aid or detract from the satisfaction of another, and the environment can increase or reduce requirement satisfaction. These kinds of conflicts have been addressed in goal-oriented requirements analysis [25] and in non-functional requirements engineering (NFR) [12].

In the security community, the integration of stakeholder views has lead to the paradigm of *multilateral security* [7], which acknowledges that different stakeholders have different, but equally justified security goals. Similarly, different types of requirements may interact. Therefore, it is necessary to consider all views for each type of requirement, i.e., functional and non-functional requirements, to come up with a consistent set of *system requirements*, shown at the center of Fig. 1. The figure shows two steps of requirements reconciliation: in the first, the stakeholder views for requirements of the same classes are reconciled. In particular, the security stakeholder views are reconciled to a set of *security system requirements*. In the second step, interacting requirements of different types are reconciled to come up with a consistent set of *system requirements* that contains requirements of all types.

The reconciliation process may require stakeholders to compromise on their initial requirements. For example, the requirement of keeping the balance of a bank account confidential against all other customers contradicts the fact that a bank transfer (which would be described in the functional requirements) necessarily leaks information about the account of which the transfered amount is withdrawn. Therefore, the customer may relax the confidentiality requirement and be content with restricting the precision with which other customers can estimate the balance.

2.4 Specification and domain knowledge

The system requirements describe properties the system must have after the machine has been built. They do not prescribe how the machine or the environment contribute to achieve such a system. Therefore, the system requirements are refined into a machine *specification*, as well as *domain knowledge*, consisting of *facts* and *assumptions* about the environment [8, 26–28]. Conjoined, those three sets of properties (specification, facts, assumptions) must be sufficient to satisfy the system requirements, indicated by the dashed arrow in Fig. 1.

The specification constrains the machine to be built. The facts and assumptions describe or constrain the environment of the machine. This distinction is particularly important for security requirements. A machine usually cannot satisfy security requirements unconditionally. It can provide security mechanisms that contribute to system security, but cannot enforce system security on its own.

In the example, the specification will prescribe access control mechanisms to prevent unauthorized access to bank account information. However, access control can only contribute to achieving the requirement that other customers do not get to know the balance of an account. It is furthermore necessary to know that there is no physical access to the servers of the bank (a fact), and to assume that no customer can collect information about all bank transfers related to a particular account and thereby restrict the probable balance of that account beyond the limits that the security requirement permits.

At the design level, not only the design of the machine must refine the specification, but also additional facts and assumptions need to be considered. Here, trust is relevant. Often, stakeholders cannot enforce their requirements alone but need to delegate tasks to other actors in the environment of the machine. Consequently, they need to trust those actors to accomplish the tasks in a secure way. For example, if the machine design stipulates that administrators can override access control mechanisms, then the ordinary users necessarily need to trust system administrators to only use their rights securely.

At the implementation level, assumptions are refined to organizational procedures and processes that prescribe how the implemented machine must be used in order to achieve security.

2.5 Threat analysis concepts and the concretization process

Figure 1 describes security requirements engineering primarily from a software engineering perspective. On the other hand, the right-hand side of Fig. 2 presents a complementary view, emphasizing concepts from security engineering and management, such as threats and risks, see for example, the Common Criteria [1] and Fig. 12. Those threat or risk analysis concepts are used by some of the SRE approaches discussed later in this paper (see Sect. 8).

From a threat analysis perspective, a stakeholder requires a *security property* to hold for a *resource*, whose violation implies a potential *loss* to the stakeholder. This violation can be caused by a *vulnerability*, which could potentially be exploited by a *threat* initiated by a *threat agent*. An *attack* actually exploits a vulnerability, and is initiated by an *attacker*. Attackers are a subset of threat agents (often also called *adversaries*), which in turn constitute a subset of the *counter-stakeholders* discussed in Sect. 2.2. The potential loss constitutes a *risk* for the stakeholder, but can be reduced by *countermeasures* mitigating the vulnerability.

How does this perspective relate to the concretization process of SRE depicted in Fig. 1? The main insight refers to the nature of the security property, which can embody various abstraction levels. The left-hand side of Fig. 2 shows examples of security properties, ordered by their level of abstraction. Corresponding to the concretization process described in Fig. 1, the highly abstract stakeholder goals become more concrete during the requirements engineering process. A *goal* is a security property of an asset, in which the stakeholder is interested. Goals get more detailed by transforming them into *requirements*, and from there to *system requirements*. Those get more concrete by the conjunction of specification and assumptions (supported by facts). A *security specification* is a property that the machine must satisfy in order to achieve a security requirement. An *assumption* is a security property addressing the same level of abstraction as a specification. A fact itself, however, is *not* a security property, because facts are true without any precondition (otherwise, they would be assumptions). This process leads to design, implementation, and usage properties that are the most concrete representations of the abstract goals.

At the same time, the *resource*, to which the security property refers, becomes less abstract during the process. As Fig. 1 shows, the resource of a security goal is an asset, and the resource of a security requirement is a piece of information. The resource of a low-level property such as a design property may be a data record or a communication protocol between components of the machine.

An important point is that all abstraction levels of security properties can be subject to *threats* imposed by *threat agents*. This allows for an integration of iterative threat analysis processes into the likewise iterative processes of requirements engineering. At multiple concretization levels, threat analysis processes could produce insights for the requirements engineering process, for example, if countermeasures are integrated into a new version of functional requirements.

However, the consequences of threats, e.g. a potential loss, and the effects of countermeasures on the security of the system cannot be evaluated uniformly for all abstraction levels of security properties. For example, a stakeholder can assign a *potential loss* to a security goal or a security requirement, and thus (quantitatively or qualitatively) express the value of that security property. To assign a loss to security properties at a high level of abstraction can be useful to justify compromises when reconciling conflicting requirements. However, it is hardly possible to directly assign a loss to a security property of the machine or the environment at a lower level of abstraction. The consequences of violating such a property for the security of the system as a whole cannot be determined without knowing to what high-level security properties a low-level property contributes. Therefore, the threats and countermeasures for low-level properties cannot be evaluated per se, but need to be related to the corresponding high-level properties. In general, the exact relationships between security properties at different levels of abstraction must be maintained; it should also be established how low-level threats affect higher-level security goals or requirements [29].

3 Related work

Our conceptual framework is comparable to the work of Moffett et al. [30] that defines core security requirements artifacts. Their framework is intended to eventually derive an SRE process rather then to facilitate the comparison of different security requirements engineering methods. Hence, the authors use certain definitions of concepts while deliberately leaving others out, and define dependencies such that concrete process steps can be stabilized.

In the framework of Moffet et al., an artefact is defined as any object created as part of the process of system development: starting with documents and prototypes all the way down to the working system itself. A distinction is made between core and supportive artifacts. Core requirements engineering artifacts consist of goals, requirements, and the components and structure of the system, while core security requirements engineering artifacts consist of assets, threats, and control principles.

Our understanding of functional and security goals overlaps with that of Moffet et al. In comparison, the definition of security requirements as constraints is too restricted for our purpose: our objective is to keep the conceptual framework general enough to account for other approaches and hence enable comparison between the different methods.

The control principles mentioned by the authors, e.g., separation of duties or principle of least privilege, we interpret as design principles rather than as core security requirements artifacts. We do have a comparable "(organizational) procedures and processes" concept (see Fig. 1) to accommodate some of the control principles. We agree that considering organizational principles from the beginning may be useful, as suggested by the authors and also by [11, 31], if an organizational setting is the starting point of analysis.

The conceptual framework is also similar to the taxonomy provided by Firesmith [5], which is based on [30]. We use this taxonomy as a comparative model for the main security concepts in our conceptual framework. Note that [5] actually concludes with an information model of defensibility engineering—including safety, security, and survivability quality factors. None of the methodologies that we study conflate these three fields. Our focus hence remains on security engineering, and we, therefore, compare our conceptual framework to the "information model for security engineering" given in the same paper. In our conceptual framework, other "quality factors" besides security are subsumed under the title other "non-functional requirements". We note this as an important area to explore in future research.

The conceptual framework is different from the taxonomy in [5], since it is not only about static relationships, but also includes references to reconciliation or validation activities that need to be executed to move from one concept to the other. At the same time, since the conceptual framework is not a universal process model for SRE, the activities can occur in different orders throughout the development process. The emphasized activities and the order in which they are executed is likely to depend on the emphasis of the chosen method on security engineering (driven by assets and risks) or software engineering (driven by stakeholder needs and stepwise development). The dependencies can also be traversed in different directions, e.g. in the example of validating the fulfillment of system requirements. Further, the concepts we study may change depending on the abstraction level, and hence are not easily mapped to relationships on one level of abstraction as is the case with [5]. The importance of abstraction levels is discussed in Sect. 2.5.

All concepts in the information model of security engineering have a correspondence in our conceptual framework. Nonetheless, these are not always one-to-one correspondences. In Table 2, we compare the concepts in the "information model" and the concepts in our conceptual framework. If we have differing definitions for the concepts these are mentioned in the table.

Our concepts stem from literature analysis of terminology as used in different security standards, security requirements engineering methods, and the occasional need to emphasize the differences in meaning between our concepts and existing security concepts. For example, although the importance of the different views of the stakeholders is mentioned in [5], the information models never include stakeholders. People in the information models only appear as those who may be subject to harm. In our CF, we emphasize multilaterality and hence point to the difficulty of determining what should count as an asset, and of what value it is to whom.

Another comparable study emphasizes the need for an alignment of security engineering concepts [32, 33]. The authors' objective is to align concepts in information systems risk management methods, to analyze which concepts are supported by existing security requirements engineering methods, and to define a language with solid conceptual foundations based on the prior activities. Similar to [5], the language used in [33] to model core security concepts is based on UML class diagrams and models the concepts at a

Table 2 Comparison of concepts in the information model for security engineering [5] and our CF

System	The system in [5] refers to the machine and the people that interact with the machine. In the CF, the system is the machine and the environment. It is less machine centric. In the case of risk and threat analysis, we talk of system security properties that are breached, rather than the states of the system.
Environment	The environment in [5] is the physical environment, which may be subject to harm. In the CF it is where the machine is embedded and plays an important role in the definition of the functionality and security requirements of the system.
Asset	... is an asset in CF. However, we take a multilateral approach and do not consider assets only to be threatened by malicious attacks. In order to distinguish assets in the different abstraction levels, generically we call it resource.
Security goal	In the CF, security goals are called the same and refer to the security goals defined towards the assets important to the different stakeholders.
Security policy	In [5], security policies mandate security criteria. We refer to a similar phenomena in a specification, which mandates all the security requirements that need to be included in the design of the system. Further, we assume that there are security policies defined in the organizational procedures and processes, which mandate protection of security within the environment.
Security requirement	Same definition
Threat	Same definition
Attack	Same definition
Attacker	... is in the CF called attacker in the case of actual exploitation, or threat agent if the exploitation is potential. The concept of counter-stakeholder generalizes both concepts.
Harm	... is called a loss.
Security mechanism	... is called a countermeasure.
Property	... are subsumed under loss.
People	
Service	

Table 3 Comparison of concepts in the information system security risk management domain model [32] and our CF

Asset	... is an asset in CF. In contrast to the definition in the CF that involves multilaterality, the definition by Mayer only involves malicious attacks. Mayer refers to an asset as a general concept similar to what is called resource in the CF. Concrete assets are business assets (e.g., information, processes, skills) and information system assets (i.e., a part of the information system that has value to the organization).
Security criterion	... is called a security goal.
Vulnerability	Same definition
Threat	Mayer defines a threat as a composition of a threat agent and an attack method. The latter represents the means used to carry out a threat. Our CF does not explicitly mention attack methods. However, they are subsumed under the term threat.
Threat agent	Same definition
Security requirement	Mayer's definition is similar to our definition with the difference that our definition does not (directly) refer to risk. In our CF, risk is not considered on the low levels of abstraction, i.e., risk comes into play when the security property can be considered on a level of abstraction higher than that of system requirements (cf. Sect. 2.5).
Control	... is called a countermeasure.

fixed level of abstraction. It does not address different levels of abstraction as necessary in a software engineering approach. We include and extend the core security concepts defined in [32, 33] in our conceptual framework.

In Table 3, we compare the concepts in the information system security risk management domain model presented in [32] and the concepts in our conceptual framework. If we have differing definitions for the concepts, these are mentioned in the table.

Further, a survey of SRE methods has also been provided in [34], but the review predominantly focuses on methods for risk management and analyzes them with respect to their compliance to the Common Criteria. In [35] and [36], the authors provide two short surveys of security requirements engineering methods, but the analysis in both articles is limited in scope and in detail. In addition, they lack a reference like the conceptual framework through which detailed comparisons can be conducted.

Recently, an critique has been raised by Jureta et al. [15] with respect to the terminology by Jackson and Zave that states that the requirements problem amounts to finding the specification and domain assumptions that suffice to satisfy the requirements. The authors argue that this model does not facilitate alternative articulations of domain assumptions, specifications and requirements, and does not capacitate the stakeholders to make preferences between these alternatives.

According to the core requirements engineering ontology by Jureta et al. [15], functional requirements describe what the system does, while non-functional requirements how well the system does it (i.e., quality requirements). Non-functional requirements are then divided into two: those which provide measurable "objective" qualities and structured quality values (quality constraints), and those which provide subjective and unstructured or ill-defined quality values (called softgoals). Softgoals are not satisfiable but "satisficable" by justifiably approximated quality constraints.

The authors add that there will be multiple subsets of domain assumptions and specifications that will fulfill the requirements. In order to deal with these alternatives, they suggest determining which of the goals and quality constraints are optional or mandatory. This dichotomous classification can then be used to facilitate the negotiation of stakeholder preferences between different solutions to the problem.

Our description of the concretization process in Sect. 2.2 explicates some of the aspects of what the authors in [15] call justified approximation e.g., refining information that a goal refers to, stakeholders, counter-stakeholders, circumstances, etc. Section 2.5 relates how the different security concepts like harm, risk, and threats apply at the different levels of abstraction to these aspects. In contrast to the authors, we do not assume that all non-functional goals will only lead to approximating quality constraints, but may also lead to additional or modified functional requirements, as also described in [30] and [37].

In the following Sects. 4–9, we present the survey of existing SRE approaches based on our conceptual framework. We start with multilateral approaches, followed by UML-based and goal-oriented methods, approaches using problem frames, risk-oriented, and Common Criteria-based methods. For each of the selected approaches, we describe the method, its *scope* in terms of the system development tasks and security goals it covers, the *validation and quality assurance* aspects it entails, and the *relation to our conceptual framework* concerning the nomenclature used by the methods.

4 Multilateral approaches

4.1 Multilateral security requirements analysis (MSRA)

(1) Description: The objective of the Multilateral Security Requirements Analysis (MSRA) method [38, 39] is to

apply the principles of multilateral security [7] during the requirements engineering phase of systems development. This is done by analyzing security and privacy needs of all the stakeholders of a system-to-be, identifying conflicts, and consolidating the different stakeholder views. The method borrows both from theories on multilateral security and viewpoint-oriented requirements engineering.

In order to articulate the different security needs of the stakeholders, MSRA users elaborate security requirements from the perspectives of the different stakeholders with respect to bundled functionalities of a system. Security requirements result from the reconciliation of multilateral security goals. Security goals are selected from a rich taxonomy derived from the CIA triad, which also includes properties such as accountability and pseudonymity etc. Security goals, and later requirements, contain the attributes *stakeholders* who have an interest in the requirement, *counter-stakeholders* towards whom a requirement is stated, and a number of other attributes that are defined in the following paragraphs.

A stakeholder is defined as any person or organization that has an interest in the system-to-be. Therewith, the elaboration of the security requirements is not limited to the functional users of the system-to-be, the latter being referred to as *actors*. Rather, a distinction is made that allows the elaboration of both, those who have a stake in the system security, and those who will be using the system.

The variant *Confidentiality Requirements Elicitation and Engineering* (CREE) of MSRA [40] considers only confidentiality requirements. Later work has focused on the formalization of the confidentiality requirements in CREE and the use of defeasible logic[6] to analyze ambiguities and conflicts [41]. *Counter-stakeholders* refer to those stakeholders whom the security goals are directed at. These may or may not be malicious attackers or actors of the system.

Further, MSRA works with an *information model*, the elements of which are the objects of the different security requirements. The information model is of a higher level of abstraction than a data model, as would be necessary for a functional specification of the system-to-be.

Additional attributes of a security requirement are: the *owner* of the security requirement; the *degree of agreement* among stakeholders towards the security requirement; the *goal* of the requirement (in CREE this is only confidentiality or consent); the *information* the requirement addresses; the *strictness*, stating if the security requirement makes a statement about the security of information that it is not explicitly addressing; and the *rationale*, articulating why the information needs to be secured. Further, *temporal validity*, defining how long the security concern must be preserved, is seen as an attribute, but is not handled in the tables.

An *episode* comprises system functionality that relates to similar security interests of a single or a group of stakeholder(s). They are useful for identifying conflicts between the security goals. Several kinds of conflicts between security goals can exist: for a single stakeholder, between the different requirements she has towards multiple episodes; between the different stakeholders of an episode; and between the requirements of episodes regardless of stakeholders.

Once the conflicts and inconsistencies are addressed, the security goals are said to be refined into security requirements. Further, additional conflicts may exist between security requirements, functional requirements, and other non-functional requirements. The method proposes analyzing conflicts carefully and solving them either during requirements analysis, through design, or using negotiation mechanisms at runtime.

The following are the main steps of the multilateral security requirements analysis method, once an initial functional requirements analysis for the main functionalities of the system is concluded:

1. Identify stakeholders: Stakeholders are all parties that have functional, security, privacy, or information interests in the system-to-be.
2. Identify episodes: Episodes are similar to scenarios, but are of a lower granularity, identifying sets of functionalities as would be meaningful to users. Episodes are used to partition the security goals and are later useful in identifying conflicts between multiple security goals.
3. Elaborate security goals: Identify and describe the security goals of the different security stakeholders for each of the episodes.
4. Identify facts and assumptions: These are the properties of the environment that are relevant for stating security goals.
5. Refine stakeholder views on episodes: Elaborate the stakeholder views taking facts, assumptions, and the relationships between episodes into account.
6. Reconcile security goals: Identify conflicts between security goals, find compromises between conflicting goals, and establish a consistent set of security system requirements.
7. Reconcile security and functional requirements: Trade functionality for security and vice versa in case of conflicting functional and security requirements.

(2) Scope: MSRA is integrated into the requirements analysis phase and can be applied as soon as the initial functional requirements of the system are identified.

All CIA goals are considered, although the emphasis on privacy has put the focus on confidentiality and integrity

[6] A non-monotonic logic in which defeasible rules can be overridden by others when certain conditions hold. For example, in the case of an emergency, certain confidentiality rules can be overridden.

Table 4 Example requirements table in MSRA/CREE

Id	Own.	Degree agree.	Goal	Counter-stakeh.	Strict.	Info	Context	Ration.
E.1	Patient	Unanim.	Consent	Clinician	Strict	PII	New patient	Admin/care
E.2	Patient	Partial	Confid.	Govern.	Non-strict	PII	Aggregate data	Anon

goals. MSRA puts an emphasis on multilateral security, focusing on stakeholder views, the circumstances of security requirements, and reconciliation of conflicting requirements. MSRA uses UML models to capture episodes, the information model, as well as the functional and security requirements. In CREE, tables are used to denote the attributes of security requirements as shown by the example in Table 4.

(3) Validation and quality assurance (QA): MSRA does not provide explicit validation methods. It does not guarantee completeness of security requirements, although multilateral security is an attempt to define security and privacy requirements in a system for all stakeholders, with the intention of discovering requirements that from a monolithic technical perspective could else have been missed.

Through the multilateral view to security, conflicts are a central concern of the method. The method explicitly addresses interactions among security requirements, as well as between security and functional requirements. Later formalization work with defeasible logic proposes automated analysis of the requirements for conflicts and ambiguities. Non-functional requirements other than security are not considered. The method is iterative, in the sense that after the reconciliation of security goals into security requirements, interactions are expected to affect the functional requirements of the system, requiring a review of the security requirements.

(4) Relationship to the conceptual framework: The stakeholders and counter-stakeholders in MSRA correspond to the same terminology in our conceptual framework. The information model captures the information associated with a security requirement in Fig. 1. Security goals refer to goals as in the conceptual framework, whereas security requirements refer to the security system requirements. Episodes map to circumstances in the CF. Facts and assumptions map one-to-one to the same CF terminology.

4.2 Security quality requirements engineering methodology (SQUARE)

(1) Description: SQUARE [42] is a comprehensive methodology for security requirements engineering. Its aim is to integrate security requirements engineering into software development processes [43]. SQUARE stresses applicability in real software development projects and thus provides an organizational framework for carrying out security requirements engineering activities. It is assumed that SQUARE is carried out jointly by requirements engineers and stakeholders. It consists of 9 steps:

1. Agree on definitions: This step serves to enable a clear communication between requirements engineers and stakeholders.
2. Identify security goals: Initially, the stakeholders will state different security goals. In this step, the goals are aligned, and conflicts are resolved.
3. Develop artifacts: The authors name the following artifacts that should be collected: system architecture diagram, use case scenarios/diagrams, misuse case scenarios/diagrams (see Sect. 5.1), attack trees, and standardized templates and forms. These artifacts form the basis for the subsequent steps of the method.
4. Perform risk assessment: In this step, the vulnerabilities and threats related to the system are identified, as well as the likelihood that the threats will lead to attacks. The authors propose to apply existing risk assessment methods.
5. Select elicitation technique: The method selected in this step will be applied in the next step to perform the actual security requirements elicitation. Again, SQUARE recommends to apply an existing technique to be chosen for the project at hand.
6. Elicit security requirements: A crucial point in this step is to ensure that the requirements are verifiable and that they are not implementations or architectural constraints instead of requirements.
7. Categorize requirements: The elicited requirements are categorized (at least) according to the following criteria: essential, non-essential, system-level, software-level architectural constraint. Since the latter are not considered as requirements, their existence indicates that the previous steps should be executed again.
8. Prioritize requirements: It is assumed that not all requirements can be implemented; hence, the most important requirements must be identified.
9. Requirements inspection: In this last step, the requirements are checked for ambiguities, inconsistencies, mistaken assumptions, and the like. Its result is the final security requirements documents for the stakeholders.

Author's personal copy

(2) Scope: SQUARE is a comprehensive security requirements engineering methodology that recommends to make use of other techniques developed in the field, and that covers all CIA goals.

(3) Validation and QA: Each step of SQUARE closes with some exit criteria, which have to be fulfilled before the next step is begun. Moreover, the last step is exclusively dedicated to validating the requirements. However, no formal validation is performed.

(4) Relation to conceptual framework: Although SQUARE uses notions of our conceptual framework, they are often used in a narrower sense. Stakeholders are identified with clients. It seems that "system-level requirements" correspond to requirements as defined in the conceptual framework, whereas "software-level requirements" correspond to specifications. "System" seems to mean the IT infrastructure in which the software to be developed will operate. Hence, this term is also used in a narrower sense than in the conceptual framework. Domain knowledge is not mentioned explicitly. However, step 1 of the method seems to be related to it. Also, the notions asset and vulnerability are not mentioned. On the other hand, the terms security goal, threat and risk are used in the same way as in the conceptual framework.

5 UML-based approaches

In this section, we discuss approaches to security requirements engineering that make use of Unified Modeling Language (UML) [44] notation.

5.1 Misuse cases

(1) Description: Sindre and Opdahl [45] extend the traditional use case approach to also consider *misuse* cases, which represent behavior not wanted in the system to be developed. Misuse cases are initiated by misusers. A use case diagram (see Fig. 3) contains both, use cases and actors (notated as named ellipses and named stick figures, respectively), as well as misuse cases and misusers (notated as graphically inverted use cases and actors).[7]

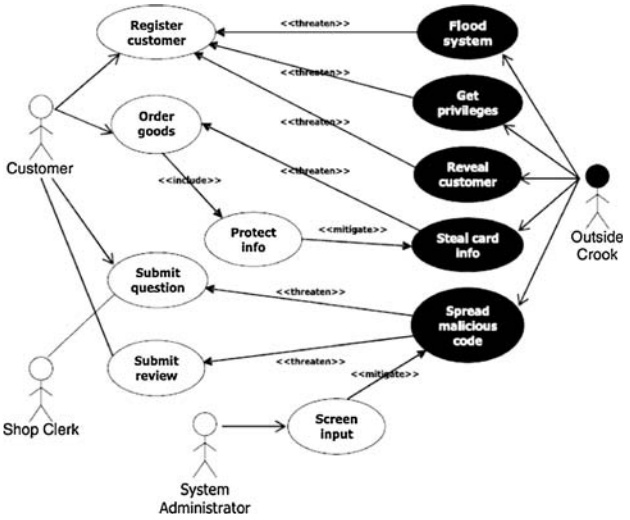


Fig. 3 Use case diagram containing misusers and misuse cases (taken from [45])

A use case is related to a misuse case using a directed association. An association pointing from a misuse case to a use case has the stereotype <<`threaten`>>. A use case diagram can contain *security use cases*, which are special use cases. An association pointing from a security use case to a misuse case has the stereotype <<`mitigate`>>. It is stated that ordinary use cases represent requirements, security cases represent security requirements, and misuse cases represent security threats. Since use case diagrams only give an overview of the system functionality, the essence of the contained uses cases is captured in an associated textual description. This textual description is based on a template to be filled out by an analyst. Sindre and Opdahl extend the template, making it suitable for describing misuse cases, supporting detailed elicitation and analysis of security threats. Furthermore, they present an iterative method based on common risk and threat analysis:

1. Identify critical assets in the system.
2. Define security goals for each asset.
3. Identify threats for each security goal by identifying stakeholders that may intentionally harm the system or its environment. Identify sequences of actions that may result in intentional harm.
4. Identify and analyze risks for the threats (using standard techniques).
5. Define security requirements for the threats to match risks and protection costs.

Applying misuse cases results in a use case diagram including use cases, security uses cases, and misuse cases. The approach neither considers a formal foundation nor an attacker model.

(2) Scope: Misuse cases are applicable to *design* a system that covers different security needs. It is possible to consider all three CIA goals. It incorporates common risk and threat analysis techniques.

(3) Validation and QA: The interaction between functional and security needs is considered in terms of linked use cases and security use cases in a use case diagram. The approach does not consider elicitation of requirements (in the sense of security requirements in the conceptual framework),

[7] The definition of mal-activity diagrams [46] is based on a similar idea. In this recent work, malicious activities and actors are added to UML activity diagrams in order to model potential attacks.

completeness of the set of requirements, validation, verification, conflicting requirements, nor the interaction of security and other non-functional requirements.

(4) Relation to conceptual framework: Sindre and Opdahl state that a *critical asset* is "either information that the enterprise possesses, virtual locations that the enterprise controls, or computerized activities that the enterprise performs". Hence, their definition is close to the ISO/IEC 13335-1 definition: anything that has a value to the organization. Their definition of a security goal is vague and refers to the "security criteria" of the Common Criteria. What they call "stakeholder that may intentionally harm..." is represented by the notion of a counter-stakeholder in the conceptual framework. Misuse cases use the terms actor and stakeholder synonymously. They only describe a narrower view of the notion stakeholder of our conceptual framework.

The notions threat and risk of the misuse case approach can be mapped to the equally named notions of our conceptual framework. Furthermore, the authors call security requirements what according to our conceptual framework are specifications (or even design solutions) to the mentioned security goals. The notions security requirement, domain knowledge, and vulnerability as defined in our conceptual framework are not considered by the misuse case approach.

5.2 SecureUML

(1) Description: Lodderstedt et al. [47] present a UML-based modeling language for the development of secure, distributed systems called *SecureUML*. In particular, their approach focuses on embedding role-based access control policies in UML class diagrams using a UML profile. The UML profile defines a vocabulary for annotating class diagrams with relevant access control information. Furthermore, authorization constraints in terms of OCL [48] preconditions are developed. They make it possible to formally express role-based access control policies for certain class components. SecureUML does not consider an attacker model.

(2) Scope: Lodderstedt et al. focus on the *design* of role-based access control policies, a rather partial mechanism to fulfill confidentiality and integrity goals. Availability is not covered by this method.

(3) Validation and QA: SecureUML does not consider requirements (in the sense of security requirements in the conceptual framework) elicitation, completeness of the set of requirements, validation or verification, nor interaction and conflicts of requirements.

(4) Relation to conceptual framework: The definition of the notion security requirement in SecureUML matches the definition of the notion specification of our conceptual framework presented in Sect. 2. SecureUML can be considered as a notation to specify and design secure software systems, rather than a security requirements engineering method. SecureUML deals with users, which can be considered as stakeholders in our conceptual framework. The notions security goal and requirement, domain knowledge, asset, threat, vulnerability, and risk as defined in our conceptual framework are not considered by SecureUML.

5.3 UMLsec

(1) Description: Jürjens [49] introduces a UML-based modeling language for the development of security-critical systems named *UMLsec*. His approach considers several security requirements according to the CIA triad. These requirements are depicted in different UML diagrams using stereotypes, constraints, and tagged values, which are defined in a UML profile. The UMLsec extensions are precisely defined and have a formal semantics. Jürjens' work considers an attacker model based on the *adversary* tag. The approach also considers domain knowledge in terms of assumptions.

(2) Scope: Jürjens' approach focuses on the *design* of a machine, and it covers all three CIA goals.

(3) Validation and QA: Jürjens states that the formal foundation makes it possible to apply traditional verification techniques. For this purpose, a tool suite is provided. UMLsec does not consider elicitation of requirements (in the sense of CF security requirements), completeness of the set of requirements, verification, conflicting requirements, nor possible interaction of security, functional, and other non-functional requirements.

(4) Relation to conceptual framework: The definition of the notion security requirement in UMLsec matches the definition of the notion specification of our conceptual framework presented in Sect. 2. Hence, similar to SecureUML, UMLsec can be considered as a notation to specify and design secure software systems rather than a security requirements engineering method. The notions stakeholder and domain knowledge of our conceptual framework are partly covered by the UMLsec notions actor and assumption, respectively. The UMLsec notions threat, vulnerability, and risk can be mapped to the equally named notions of our conceptual framework. The notions of security goal and requirement as well as asset are not considered by UMLsec.

6 Goal-oriented approaches

6.1 Keep all objectives satisfied (KAOS) with intentional anti-models

(1) Description: In his paper on "Engineering requirements for system reliability and security" [37], van Lamsweerde

pulls together all his previous research on the goal-oriented requirements analysis method KAOS [50], formalization of requirements using linear time temporal logic [51], requirements conflict analysis [25], and the use of anti-models for elaborating security requirements [52]. Therefore, van Lamsweerde does not suggest a method specifically for elaborating security goals, but extends KAOS to include the elaboration of security requirements.

KAOS takes into consideration that there are multiple stakeholders in and multiple views towards a system-to-be. The views here do not refer to the differing views of the stakeholders, but to the goal, object, agent, system operation, obstacle, security-threat and agent behavior models—each model stands for a different view of the system. In the goal model, the goal of a stakeholder is refined using an AND/OR refinement tree by asking the questions why (for upward elaboration) and how (for downward refinement) to the leaves of the goal-refinement tree. These leaves are then assigned to the agents. A requirement is a goal assigned to a single agent in the software-to-be. It is acknowledged that conflicts can exist among the goals of the different stakeholders, and these conflicts are managed—detected, analyzed, and resolved—using a number of heuristics. The integration of the multiple views is performed systematically by stepping through the following activities:

1. Domain analysis, part 1: consists of a goal model of the current system-as-is.
2. Domain analysis, part 2: consists of deriving an object model of the system-as-is.
3. System-to-be analysis: replay of Step 1 for the system-to-be.
4. System-to-be analysis: replay of Step 2 for the system-to-be.
5. Obstacle and threat analysis: consists of building obstacle and threat models and exploring resolutions to enrich and update the goal model.
6. Conflict analysis: consists of detecting conflicts among goals and exploring resolutions to enrich and update the goal model.
7. Responsibility analysis: exploring alternative assignments of leaf goals to system agents, selecting best alternatives based on non-functional goals from the goal model, and building an agent model.
8. Goal Operationalization: build an operation model ensuring that all leaf goals from the goal model are satisfied.
9. Behavior analysis: is about building a behavior model for the system as a parallel composition of behavior models for each component.

The steps of the KAOS method are re-iterable and cyclic, although they contain data dependencies. In [52], the authors describe the elaboration of security requirements in more detail. An anti-model is constructed after the goals of the system-to-be have been elaborated and refined. This is done through the execution of the following steps:

1. Obtain the initial goals by negating existing security goals (roots of the anti-goal refinement trees).
2. Elicit potential attacker agents.
3. Perform an AND/OR refinement of anti-goals. When the anti-goal refinement trees are generated, assign anti-requirements to attackers and vulnerabilities to attackers.
4. Derive the object-agent anti-model.
5. AND/OR operationalize all anti-requirements.

The first activity includes not only the generation of obstacles through negating existing goals, but also the use of logical techniques to capture obstacles. These obstacles are then refined using fault trees, whose roots are the goals and leaves are vulnerabilities, into an obstacle model. Such a model shows how security goals can be obstructed by linking negated goals to the attacker's malicious goals, called anti-goals, and capabilities. These capabilities define the interface between the attacker and its own environment, including the threatened software-to-be. The properties of the attacker's environment comprise the properties of the software-to-be, including vulnerabilities that can be exploited for anti-goal achievement. The elicitation of potential attacker agents and the refinement of the obstacle model are completed in steps two and three respectively.

Once an anti-model stands and the resulting obstacles have been identified, the requirements engineers are expected to develop countermeasures so that the preconditions of the anti-goals are no longer fulfilled. Countermeasures are selected based on (a) the severity and likelihood of the corresponding threat, and (b) non-functional goals that have been identified prior to the anti-model. Alternative countermeasures can be produced using goal substitution, agent substitution, goal weakening, goal restoration, and anti-goal mitigation.

All requirements in KAOS are written by default using semi-formal graphical notations and, if needed, using formal notation. This also holds for the anti-goals and security threats posed by an attacker. A linear real-time temporal logic is used to formalize the goals, domain properties, and required trigger conditions. A simple state-based *Z*-like language is used to express preconditions and postconditions.

(2) Scope: KAOS is a requirements engineering method concerned with the elaboration of the objectives to be achieved by the system-to-be, the operationalization of such objectives into requirements and assumptions, the assignment of responsibilities for those specifications to agents such as humans, devices or software, and the

evolution of such requirements over time and across system families. Nevertheless, we observe that the method focuses less on the elicitation of requirements, but more on the completeness, consistency, and feasibility of requirements as well as their successful operationalization in specifications.

(3) Validation and QA: The completeness, consistency, and validation of the elaborated requirements are an important objective of the KAOS method and its formalization mechanisms. Goals provide a criterion for completeness. A goal is fulfilled if it is satisfied by the requirements in view of the domain properties and under certain expectations. Obstacle analysis and the derived countermeasures are also used to reach goal completeness.

The method also suggests a number of activities for the verification of requirements. A first kind of verification consists in checking that the refinements of non-soft goals in the goal model are correct and complete so that missing subgoals can be avoided.

A second approach to model verification consists of checking the correctness of operationalizations of goals from the goal model into specifications of operations from the operational model. Formal methods and formal refinement patterns are offered to execute both types of verification.

Further, consistency is addressed through the analysis of conflicts among multiple requirements.

(4) Relation to conceptual framework: The terminology of the KAOS framework maps to the terminology of the conceptual framework as follows. Goals refer to functional and non-functional goals. Objects refer to information or system components. Agents refer to stakeholders with functional assignments or functional system resources. Obstacles refer to either conflicting goals or requirements of the different stakeholder views in the conceptual framework, or to goals that conflict with the security properties as defined in Fig. 2. Security threats are the threats as posed by threat agents. All the analysis that KAOS proposes with respect to the operationalization of goals refers to the consistency, validation, and completeness analyses of the specification of the system, namely, analyses that occur after the system requirements have been elaborated.

The KAOS method considers also domain properties and expectations, which correspond to facts and assumptions in the conceptual framework. As a result, threat analysis encompasses an analysis of the environment.

6.2 Secure i^* and Secure Tropos

Tropos is a software development methodology based on the paradigm of agent-oriented software development [53–55]. Tropos deals with all analysis, design, and implementation activities in a software development process, with a strong focus on the early phases of software development. Tropos incorporates many of the concepts of Yu's i^*-modeling framework [56–58]. For that reason, the following description of Tropos also applies to a large extent to the i^*-modeling framework[8].

Models in Tropos are instances of a metamodel [54]. This metamodel consists of the following concepts and relationships:

- *Actor*: An actor is an entity that has goals within the system or the organization of interest. This can be a physical or a software agent, as well as a role (an agent can play a role) or a position (a set of roles, a position covers roles).
- *Goal*: Goals represent actors' interests towards the system. Tropos distinguishes hard goals and soft goals. Hard goals describe conditions that an actor would like to achieve. Soft goals have no formal, clear-cut definition or satisfaction criteria. Soft goals are typically used to model non-functional requirements [59, p. 4].
- *Plan or task*: an abstraction of doing something. If the task has dependencies to the machine, it refers to the specification.
- *Resource*: represents a physical or informational entity. If the resource has dependencies to security goals it refers to an asset (respectively, to information).
- *Dependency*: Dependencies model the fact that one actor depends for some reason on another to attain a goal, execute some plan, or deliver a resource. Thus a dependency is a ternary relationship between a depender, a dependee and a dependum.

The Tropos concepts are graphically represented as shown in Fig. 4.

Tropos distinguishes five main development phases: *early requirements*, *late requirements*, *architectural design*, *detailed design*, and *implementation*. During the development process, the models are incrementally refined and extended until executable development artifacts emerge. The early requirements phase is concerned with analyzing and understanding the organizational context. During this phase the actors are identified and modeled. Actors have goals and depend on each other to fulfill goals, perform tasks, and to furnish resources. Next, during the late requirement phase, the system-to-be is described within its environment. This is done by representing the system-to-be as a number of actors who have dependencies to other elements of the model. These dependencies define the machine's functional and non-functional requirements. Architectural design is concerned with the overall structure

[8] Details about the i^*-modeling framework can be found online: http://www.istar.rwth-aachen.de/

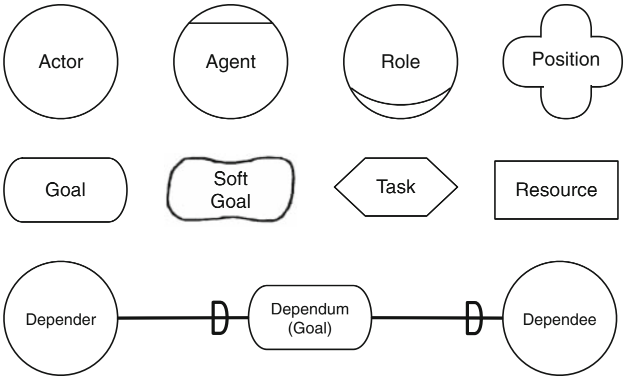


Fig. 4 Graphical representation of Tropos concepts (cf. 60)

of the machine. Subsystems and system components of the machine are represented as actors, too. Each architectural component is further detailed in terms of inputs, outputs, and control. Finally, the implementation phase comprises the mapping of the detailed design to the implementation platform.

There exist two extensions of Tropos called Secure Tropos: one by Mouratidis et al. [60–62], and another one by Massacci et al. [63]. Furthermore, there exists an extension of the i^*-modeling framework called Secure i^* [64].

(1) Description:

(a) Secure Tropos by Mouratidis et al.: Mouratidis et al. extend the Tropos methodology with new concepts to cover security modelling:

- *Security constraint*: A security constraint is defined as "a restriction related to security issues, such as privacy, integrity and availability, which can influence the analysis and design of a multiagent system under development by restricting some alternative design solutions, by conflicting with some of the requirements of the system, or by refining some of the system's objectives." [60]. They are graphically represented as clouds that are labeled with a constraint.
- *Secure dependency*: A secure dependency describes one or more security constraints that must be fulfilled for a dependency to be satisfied: "... the depender expects from the dependee to satisfy the security constraint(s) and also that the dependee will make an effort to deliver the dependum by satisfying the security constraint(s)." [60].
- *Secure entity*: A secure entity represents any secure goal/task/resource of the system.

The Secure Tropos concepts are graphically represented as shown in Figs. 5 and 6.

The Secure Tropos process is similar to the earlier-mentioned Tropos process, but is extended with phases to analyze and model the new concepts. These activities produce different kinds of diagrams, which are used as input to the later activities.

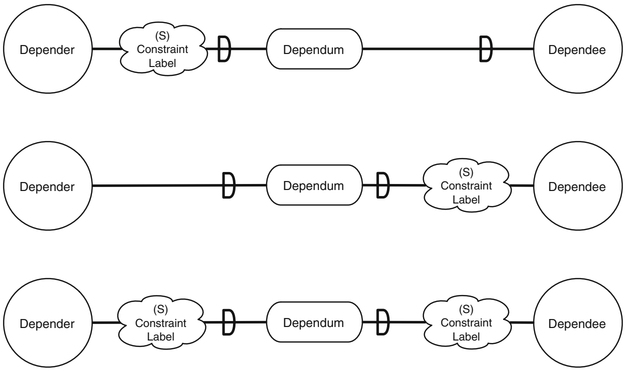


Fig. 5 Graphical representation of secure dependencies (cf. [60])

Security reference modeling deals with the security features of the system-to-be, the protection objectives of the system, the security mechanisms, and also the threats to the system's security features. These concepts are graphically represented as shown in Fig. 7, and they can be connected by two different kinds of links: a *positive contribution* link when one node helps in the fulfillment of another, and a *negative contribution* link when a node leads to the denial of another. Using these concepts and links, a developer can construct a *security reference diagram*.

Security constraint modeling covers the modeling of the security constraints, which involves the following activities:

- *Security constraint delegation*: allows the delegation of a security constraint among actors.
- *Secure constraint assignment*: indicates the assignment of a security constraint to a goal.
- *Security constraint analysis*: comprises the decomposition of a security constraint into subconstraints and the introduction of new security goals caused by the security constraint.

Secure entities modelling is an activity complementary to the security constraint modelling activity, which involves the analysis of secure goals, tasks, and resources.

Secure capabilities modelling covers the identification of the capabilities of the concerned actors and agents necessary to fulfill the security constraints.

In [65], the authors combine Secure Tropos by Mouratidis et al. with the model-based information system security risk management (ISSRM) approach by Mayer et al. [66] presented in Sect. 8.

For formal analysis, the Formal Tropos approach is inspired by KAOS [51]. For the detailed design, Agent-UML [67] is used. To analyze security attacks a scenario-based approach is used [68].

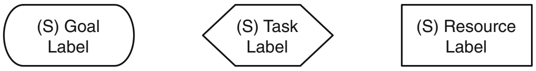


Fig. 6 Graphical representation of secure entities (cf. [60])

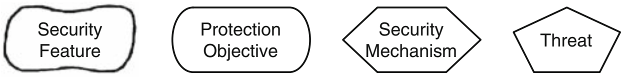


Fig. 7 Graphical representation of security reference diagram concepts (cf. [60])

b) Secure i^*: Yu et al. [64, 69] present an extension of Yu's i^*-modeling framework for modeling and analyzing security *trade-offs*. The authors argue that security measures may be in conflict with usability, performance, and functional requirements. Hence, Secure i^* focuses on the alignment of security requirements with other requirements.

Secure i^* [64] is based on a metamodel of security concepts, which considers important notions and their relationships. It is centered around *actors* that have or seek *goals*. Special *security goals* prevent or detect *threats*, or recover the system from threats. *Assets* are targeted by threats (or attacks), and actors are interested in them, actors own them, or actors delegated the usage permission of the assets to other actors. Threats occur through vulnerabilities. Threats might be unintentional, or result from accident or human error. Natural disasters are another type of threats against the systems. Most of these notions cannot be modeled using the i^* notation. Therefore, Elahi and Yu extend this graphical notation by *malicious* representations of the original i^* notions, e.g. *malicious actor*, *vulnerability*, and *threat/attack*. The graphical representation of the malicious modeling elements is similar to the i^* elements, with the difference that the malicious modeling elements have a black background. An exception are vulnerabilities. They are graphically represented by a labeled black dot at resources, and they are connected to threats/attacks via a dashed arrow pointing from threats/attacks to vulnerabilities.

Additionally, Elahi and Yu present a trade-off analysis method that makes use of the Secure i^* notation. Following their method, software engineers must balance trade-offs to mitigate threats/attacks. The denial of a goal can be analyzed and expressed through negative contribution links. Then, alternative security solutions can be examined by analyzing the impact of each of the solutions on threats/attacks and goals. Finally, a security solution that best fits with the goals of multiple actors can be selected.

(c) Secure Tropos by Massacci et al.: Secure Tropos by Massacci et al. [63] makes use of the Secure i^* (Si^*) *language*[9] (not to be confused with Secure i^* by Elahi and Yu [64]).

In addition to the notions originally supported by the i^*-modeling framework, Si^* introduces the notions of *delegation* and *trust*. Delegation is defined as a relation between two actors (the *delegator* and the *delegatee*) and a goal, task, or resource (the *delegatum*). The authors distinguish two types of delegation: delegation of *execution*, which considers the delegation of the responsibility to achieve a goal, execute a task, or deliver a resource. In contrast, delegation of *permission* considers the delegation of the permission achieve a goal, execute a task, or use a resource. The two types are graphically represented as edges between delegator, delegatee, and delegatum that are labeled with *De* (delegation of execution) or *Dp* (delegation of permission).

The notion of trust is used to separate delegation between trusted and untrusted actors. Similarly to delegation, trust is defined as a relation between two actors (the *trustor* and the *trustee*) and a goal, task, or resource (the *trustum*). Again, the authors distinguish two types of trust: trust of *execution*, which indicates the belief of one actor that the trustee will achieve the goal, accomplish the task, or deliver the resource. Furthermore, trust of *permission* indicates the belief of one actor that the trustee will not misuse the goal, task, or resource. The two types are graphically represented as edges between trustor, trustee, and trustum that are labeled with *Te* (trust of execution) or *Tp* (trust of permission).

(2) Scope:

(a) Secure Tropos by Mouratidis et al.: The Secure Tropos methodology by Mouratidis et al. can be applied in all activities in the software development process and all three CIA goals can be considered and analyzed. Furthermore, by using security attack scenarios [68], threat and attacker analysis is possible.

(b) Secure i^*: The Secure i^* methodology can be applied in all activities in the software development process. Through the security extensions of Secure Tropos, all three CIA goals can be considered and analyzed.

(c) Secure Tropos by Massacci et al.: The Secure Tropos methodology by Massacci et al. can be applied in all activities in the software development process and authorization, availability, and privacy goals can be considered and analyzed.

(3) Validation and QA:

(a) Secure Tropos by Mouratidis et al.: In order to test the developed solution, security attack scenarios are proposed [68]. A security attack scenario describes the attacker as an actor of the system, as well as the actors' goals. The aim is to test which solution could cope with different kinds of attacks. After the creation of the scenario the models are validated, e.g., by using software inspections and checklists.

[9] Details about Si^*: http://www.sesa.dit.unitn.it/sistar_tool/

(b) Secure i^*: The proposed qualitative trade-off analysis method can be seen as an informal validation procedure accomplished by assessing the impact of security solutions on the goals of actors and on threats or attacks.

(c) Secure Tropos by Massacci et al.: To automatically verify the correctness and consistency of functional and security requirements, the Secure Tropos concepts were formalized based on Datalog [70], and integrated into the CASE-Tool ST-Tool [71]. In particular, Secure Tropos assists in verification of availability, authorization, and privacy requirements and in the detection of trust conflicts [72].

Furthermore, Massacci et al. define a formal semantics of their Secure Tropos extension [63] using the *Answer Set Programming* (ASP) paradigm [73]. ASP is based on *facts* and *rules* expressed as Horn clauses. Facts are atomic statements and are used to formalize an "intuitive" description of the system. Rules can be *axioms* that are used to extend the formalization of the system description. They can also be *properties* that are used to formalize security goals as constraints. The formal foundation of Secure Tropos by Massacci et al. is sufficient to verify security goals represented as ASP properties.

(4) Relation to conceptual framework: Several notions common to all three presented approaches can be related to our conceptual framework as follows: actors partly correspond to stakeholders of our CF. In contrast to stakeholders, actors are always directly linked to the machine. Goals correspond to goals as well as requirements, since the goals in Tropos are incrementally refined. Hard and soft goals correspond to functional and non-functional requirements. Furthermore, resources correspond to the same notion of our CF.

(a) Secure Tropos by Mouratidis et al.: Security constraints and security features as well as secure entities correspond to security goals, security requirements, and specifications of our CF. The considered threats can be mapped to the threats defined in our CF. Threats, as well as vulnerabilities and risk, are covered by the combination of Secure Tropos by Mouratidis et al. and the model-based ISSRM approach by Mayer et al. [66] presented in Sect. 8. As a consequence, the notions threat, vulnerability, and risk correspond to the equally named ISSRM notions, which in turn can be directly mapped to the notions of our CF. Protection objectives describe abstract solution mechanisms such as encryption, whereas security mechanisms denote the concrete mechanisms such as AES (Advanced Encryption Standard). Secure Tropos by Mouratidis et al. does not consider domain knowledge and assets.

(b) Secure i^*: Security goals correspond to security goals, security requirements, and specifications of our CF. Assets directly correspond to assets in our CF. Secure i^* does not provide a clear-cut definition of the notions threat and attack. It seems that they can be mapped to the equally named notions of our CF. The Secure i^* notion vulnerability matches the similarly named notion of our CF. Secure i^* does not consider domain knowledge (except for distinguishing honest and malicious actors) and risk.

(c) Secure Tropos by Massacci et al.: Security constraints/properties correspond to security goals, security requirements, and specifications of our CF. Secure Tropos by Massacci et al. does not consider domain knowledge (except for distinguishing trusted and untrusted actors), assets, threats, vulnerabilities, or risk.

6.3 Goal-based requirements analysis method (GBRAM)

(1) Description: The objective of the Goal-Based Requirements Analysis Method (GBRAM) [11] is to utilize goal- and scenario-driven requirements engineering methods to formulate privacy and security policies, as well as requirements for e-commerce systems. Furthermore, the method targets change management in organizational privacy and security policies, and system requirements. Lastly the method is used to assure compliance of these system requirements to the privacy and security policies.

In a later work building upon GBRAM, He and Antòn introduce a role-engineering framework. In this "Framework for Modeling Privacy Requirements in Role Engineering" [74], goals and scenarios are adopted in order to analyze permissions and establish role hierarchies, which then can be used to define a role-based access control model (RBAC). Further, in [31], the authors suggest a context-free grammar for formalizing privacy goals articulated in natural language. The formalized goals are used to analyze and compare the system goals stated through them. We focus only on the earlier description of GBRAM as a privacy and security requirements analysis method.

Change management in policies and the business environment is accomplished through the analysis of changes made to the long- and short-term goals of the organization. *Strategic Changes* refer to long-term, broadly based initiatives, and *Tactical Changes* refer to short-term changes. Comparably, *Strategic Goals* are those that reflect high-level enterprise goals. These are useful for deriving requirements and are expected to be stable. In contrast, *Tactical Goals* are those goals that support an organization's strategic goals.

GBRAM contains a number of heuristics that can be applied to the various activities, as they are listed in Fig. 8.

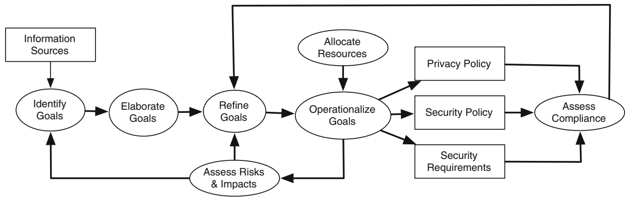


Fig. 8 GBRAM activities

The heuristics are used to identify, refine, and operationalize security and privacy goals. These are:

- *Identification heuristics* are used by the requirements analyst to study existing security and privacy policies, requirements analysis, and design documentation in order to identify both strategic and tactical goals related to the organizational assets. These goals are annotated, including information about stakeholders and responsibilities.
- *Classification heuristics* are used to classify the identified goals according to their type and dependencies.
- *Elaboration heuristics* are used to further analyze the classified goals by studying scenarios, goal obstacles, constraints, preconditions, postconditions, questions, and the underlying rationale.
- *Refinement heuristics* are used to remove synonymous and redundant goals. Inconsistencies among goals are solved, and the goals are operationalized into a "requirements specification".

Once the goals have been identified, refined and operationalized, asset-based risk assessment and compliance assessment activities are applied. These activities may result in further goal refinement or the addition of new goals to respond to the risks. In this case, the previous activities need to be reiterated to identify conflicts, avoid inconsistencies, and re-assess for resulting risks.

GBRAM is specifically useful for analyzing and elaborating organizational goals—which are already integrated into policies—to elicit system requirements. By emphasizing and integrating the management of changes in the technology and the business environment to their method, the authors manage to include important aspects that many other methods ignore.

(2) Scope: Antòn et. al. suggest using GBRAM at the beginning of the design phase in order to achieve the security of sensitive data. The heuristics are used to identify new, as well as previously overlooked, goals based on the results of risk assessment activities. The method is asset-centered and builds on the PFIRES approach for assessing risk in eCommerce systems [75]. There is no explicit mention of the use of all CIA goals. The focus in GBRAM papers is on confidentiality goals.

The GBRAM is mainly based on existing organizational policies, and hence does not provide the means to deal with different stakeholder views. The authors nevertheless emphasize the importance of reconciling conflicts among system requirements during the refinement phase.

(3) Validation and QA: Conflicts among goals are considered during the refinement activities. Although the authors state that stakeholders and their privacy concerns are important for the generation of privacy policies, no explicit reference is made as to how conflicts of interest between stakeholders can be solved using the method. Similarly, negotiation methods for solving goal conflicts are not offered, although goal conflicts and their solutions are an important part of goal-driven requirements engineering [25]. In general, no formal notation or semantics are introduced in the method.

The authors state the necessity of compliance assessment by pointing out that risk and impact assessment alone are not enough to guarantee that the system requirements are aligned with enterprise security and privacy policies. Hence, they introduce a compliance assessment activity, which is to be iteratively applied as the requirements or the policies are updated. In the compliance activity, the authors suggest identifying conflicts and inconsistencies between the policies and the requirements using the "House of Quality" approach [76]. This activity guarantees the alignment between the policies and the system requirements.

No activities are suggested for attaining the completeness of the policies or the requirements themselves. A validation of both policies and system requirements is conducted through their empirical application to multiple e-commerce Web sites. Further, the authors have collected privacy statements from over 100 commercial web sites using their Privacy Goal Management Tool in order to analyze common and conflicting goals.

(4) Relation to conceptual framework: The terminology of GBRAM can be mapped onto our conceptual framework in the following manner: security- and privacy-related statements from corresponding policies are security goals in the CF. The GBRAM definition of security requirements, which the authors call the requirements specification, corresponds to system requirements in the conceptual framework. Stakeholders can be elicited from the organizational policy documents, but no use of views is offered by the method. Requirements conflicts are considered: these are expected to occur as a result of new technologies being introduced to the application domain. Hence, stakeholder conflicts are not attended to. Asset refers to all the objects of interest centered around software, hardware, people, and documentation, whereas information refers to information assets in the CF.

7 Problem frame-based approaches

In this section, we present approaches to security requirements engineering that make use of the ideas underlying Jackson's *problem frames* [77]. Problem frames are patterns to classify software development problems.

A problem frame is described by a *frame diagram* (see Figs. 9 and 10), which basically consists of rectangles, a dashed oval, and lines between these. The task is to construct a *machine* that improves the behavior of the environment it is integrated in. The environment is described by *domains*. Jackson distinguishes between different domain types, which are depicted by differently decorated rectangles (e.g., a machine domain is denoted as a rectangle with two vertical lines).

The connecting lines between domains represent interfaces that consist of *shared phenomena*. Shared phenomena may be events, operation calls, messages, and the like. They are observable by at least two domains, but controlled by only one domain. For example, if a user types a password to log into an IT system, this is a phenomenon shared by the user and the IT system, which is controlled by the user. The *requirements* are denoted using a dashed oval. A dashed line between requirements and a domain represents a requirements reference, and an arrow pointing to a domain shows that it is a *constraining* reference.

The problem frames approach also establishes a well-formed vocabulary for the notions of requirement, specification, domain knowledge, and so on. This vocabulary constitutes the basis for the conceptual framework presented in Sect. 2. For this reason, unless otherwise noted, the vocabulary of the different approaches discussed in this section fits the vocabulary of the CF.

7.1 Abuse frames

(1) Description: Lin et al. [78] define so-called *anti-requirements* and the corresponding *abuse frames*. An anti-requirement expresses the intentions of a malicious user, and an abuse frame represents a security threat. The authors state that the purpose of anti-requirements and abuse frames is to analyze security threats and derive security requirements. Figure 9 shows an abuse frame, which constitutes a pattern to be instantiated. The instances are called *abuse frame diagrams*.

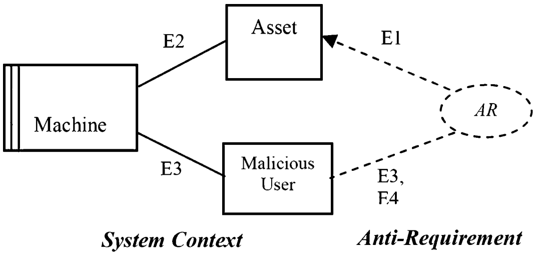


Fig. 9 Abuse frame (taken from [78])

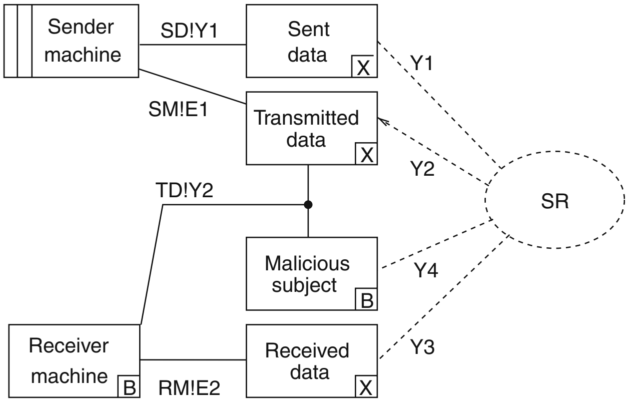


Fig. 10 Security problem frame for confidential data transmission (taken from [80])

It is stated that initially *security objectives* must be derived by identifying *critical assets* to be protected. The notion of an asset is not defined, but it seems that it is used similarly to the ISO/IEC 13335-1 definition: anything that has a value to the organization.

Based on a set of functional requirements and security objectives, the authors propose an iterative threat analysis method consisting of four steps:

1. Identify the problems and subproblems using common problem frames. Describe the security needs as constraints on the identified functionality.
2. Identify the threats and construct abuse frame diagrams. Anti-requirements are obtained by negating the security needs and capturing them in an abuse frame diagram.
3. Identify security vulnerabilities.
4. Address security vulnerabilities. It is stated that security requirements are derived, e.g. “Limit the number of tries for entering passwords”.

No formal foundation or attacker model are considered by the approach.

(2) Scope: The method is applicable to *designing* a machine. Since the authors show only a small set of the abuse frames, it is not clear if all three CIA goals are considered.

(3) Validation and QA: Abuse frames do not consider requirements elicitation, completeness of the set of requirements, validation, verification, nor interaction between security, other non-functional, and functional requirements.

(4) Relation to conceptual framework: The notion of a security objective is comparable to the security goals in our conceptual framework described in Sect. 2. Negated anti-requirements correspond to security requirements in our CF. It becomes clear that what Lin et al. call security

requirements are (according to the definition in our CF) specifications. Biddable domains can describe stakeholders and counter-stakeholders. Abuse frames consider the notions asset, threat, and vulnerability, which can be mapped to the equally named notions of our CF. In contrast, the notions domain knowledge and risk of our CF do not have a counterpart considered by the abuse frames approach.

7.2 Security engineering process using patterns (SEPP)

(1) Description: To meet the special demands of software development problems occurring in the area of security engineering, Hatebur et al. [77] introduce *security problem frames* (SPF) and *concretized security problem frames* (CSPF). SPF are special kinds of problem frames, which consider security requirements. They strictly refer to the problems concerning security, without anticipating solutions. As an example, Fig. 10 shows the frame diagram of the security problem frame for confidential data transmission. The security requirement *SR* states that the *Malicious subject* should not be able to derive *Sent data* and *Received data* using *Transmitted data*.

Solving a security problem is achieved by choosing and instantiating a CSPF. These frames are derived from the security problem frames by considering generic security mechanisms (such as using encryption for confidential data transmission).

The authors equip both kinds of frames with a formal description consisting of preconditions and postconditions [79]. These are expressed using logical formulas. The preconditions express what conditions must be met by the environment for a frame to be applicable; the postconditions are a formal representation of a (concretized) security requirement, i.e., they describe what (concretized) security requirement will be achieved by the machine to be built.

A pattern system is derived by matching the preconditions of the CSPFs with the postconditions of the SPFs.

The *security engineering process using patterns* (SEPP) [80] is illustrated in Fig. 11. Developing a secure system using (C)SPFs starts after the security goals and an initial set of security requirements are elicited. Then, each elicited security requirement must be compared to the informal descriptions of the security requirements of the SPFs (e.g., if it turns out that data must be kept confidential during its transmission, the security problem frame of Fig. 10 is applicable). After appropriate SPFs are identified for each given security requirement, these frames must be instantiated.

To instantiate those domains that represent potential attackers, a certain level of skill, equipment, and determination that a potential attacker might have must be assumed. Via these assumptions, threat models are integrated into the method.

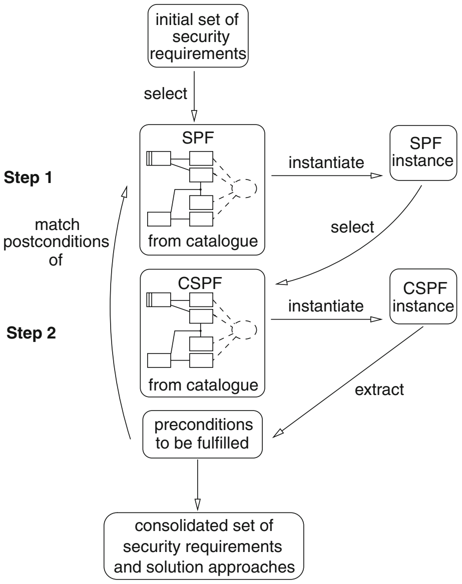


Fig. 11 Security engineering method using (C)SPFs (taken from [81])

To solve a security problem characterized by an instance of a security problem frame, the process continues with choosing a solution approach (e.g., symmetric or asymmetric encryption to keep data confidential during its transmission), thereby instantiating appropriate CSPFs. Afterwards, the preconditions of the instantiated CSPFs must be inspected. To guarantee that the preconditions hold, two alternatives are possible: either they can be *assumed* to hold, or they have to be *established* by using some security problem frame whose postconditions match the preconditions to be established.

In the second case, one must instantiate appropriate SPFs, and the earlier-mentioned procedure is repeated until all preconditions of all applied CSPFs can be proved or assumed to hold. Therefore, the security requirements engineering process will result in a set of security problems and solution approaches that additionally contains all dependent security problems and corresponding solution approaches, some of which may not have been known initially.

The authors extended their method by linking the solution approaches to *security component templates*, which can be connected to construct a secure software architecture [82]. Furthermore, formal behavioral descriptions of SPFs and CSPFs can be used to formally describe security requirements and to prove refinement relations between SPF and CSPF instances [83].

(2) Scope: Hatebur et al. have defined a pattern-based security requirements engineering method that is applicable after the security goals and an initial set of security requirements are elicited, resulting in a specification and a

software architecture consisting of component templates of a secure system. Since no frames addressing availability goals are defined so far, the method can currently be used to treat confidentiality and integrity goals.

(3) Validation and QA: The pattern system is self-contained in the sense that for any precondition of a frame covered by the pattern system, there exists at least one frame contained in the pattern system that provides a matching postcondition. Therefore, the (C)SPFs contained in the pattern system can be used to completely analyze a given security problem, whose initial security requirement is covered by one of the frames.

Since Hatebur et al. use a graphical notation including formal descriptions, tool support as well as automated validation and verification for their method is conceivable, but not yet provided. The approach does not consider the elicitation of the initial set of security requirements (only dependent security requirements are elicited), a concrete attacker model, conflicting requirements, nor interaction of security and other non-functional requirements. However, a first paper addressing the latter two issues has already been published [84].

(4) Relation to conceptual framework: The notion of security requirement matches the same notion in our CF presented in Sect. 2. The domain representing a potential attacker, such as the *Malicious subject* domain in Fig. 10 correspond to the notions of a threat agent. Furthermore, the asset or information to be protected is represented as (lexical) domains or shared phenomena. The notions of assumptions and facts (domain knowledge), specification, naturally have the same meaning as the notions in our CF. In contrast, the notions security goal, threat, vulnerability, and risk of our conceptual framework do not have a counterpart considered by SEPP.

7.3 Security requirements engineering framework (SREF)

(1) Description: Haley et al. [28, 85] present a framework for security requirements engineering. It defines the notion of security requirements, considers security requirements in an application context and helps answering the question whether the system can satisfy the security requirements. Haley et al. describe an iterative process consisting of four steps that integrates ordinary requirements engineering and security requirements engineering:

1. Identify Business (Functional) Requirements.
2. Identify security goals:
   (a) Identify candidate assets: The stated definition of an asset is close to the ISO/IEC 13335-1 definition: anything that has a value to the organization.
   (b) Generate threat descriptions: It is stated that security goals can be found by connecting the CIA concerns to the assets, which can be violated by certain actions to cause harm. Afterwards, applying prevention to the resulting threat descriptions leads to the security goals.
   (c) Apply management principles: According to the authors, such principles can be separation of duties, separation of function, etc.
3. Identify security requirements: It is stated that security requirements are constraints on functions of the system, where these constraints operationalize one or more security goals. The authors recommend to draw problem diagrams to demonstrate the functional requirements and to support capturing the security requirements in terms of constraints on the functions. The security requirements are denoted textually.
4. Construct satisfaction arguments: They show that the system can satisfy the security requirements (see "Validation and QA" for details).

The framework for security requirements engineering neither considers a formal foundation nor an attacker model.

(2) Scope: Haley et al. consider security requirements elicitation and analysis, using their security engineering framework. It covers all three CIA goals.

(3) Validation and QA: In [86], Haley et al. introduce the notion of a *trust assumption*, which is "an assumption by an analyst that the specification of a domain can depend on certain properties of some other domain in order to satisfy a security requirement". To decide whether a system can satisfy the security requirements, Haley et al. make use of structured informal and formal argumentation [87]. A two-part argument structure for security requirement *satisfaction arguments* is proposed, consisting of an informal and a formal argument. In combination with trust assumptions, satisfaction arguments facilitate showing that a system can meet its security requirements.

The framework for security requirements engineering does not consider completeness of the set of requirements, conflicting requirements (although it is mentioned that such conflicts can lead to inconsistent requirements), nor interaction between security and other non-functional requirements.

(4) Relation to conceptual framework: The notions of security goal and security requirement match the notions of our conceptual framework. Haley et al. also distinguish quality and functional goals and requirements, which matches the notions of non-functional and functional

goals and requirements of our conceptual framework. Again, the notions of (trust) assumptions and facts (domain knowledge), specification, asset, threat, and risk naturally have the same meaning as the notions of our CF. In contrast, the notion vulnerability of our CF is not considered by SREF.

8 Risk analysis-based approaches

In this section, we present approaches to security requirements engineering that are based on risk or threat analysis. The presented techniques mainly focus on the elicitation of security goals and requirements, rather than analyzing them with respect to further refinement or reconciliation of possible conflicts.

8.1 CORAS

(1) Description: CORAS [88] is a model-based method for security risk analysis. The CORAS method consists of seven steps:

1. Introductory meeting between analysts and client to clarify the client's overall goals.
2. Meeting with representatives of the client to clarify insights from first meeting and reading relevant documentation.
3. More precise description of the target to be evaluated including assumptions made.
4. Experts workshop to identify unwanted incidents such as threats and vulnerabilities.
5. Workshop to determine consequences and likelihoods of the previously identified incidents.
6. Presentation of a first risk analysis to the client to apply corrections.
7. Treatment Identification.

The artifacts produced by analysts when applying the CORAS method are denoted in the CORAS security risk modelling language [89], which is inspired by UML [44].

(2) Scope: CORAS is an organizational method that covers threat, vulnerability, and risk analysis. It also covers the elicitation of security goals.

(3) Validation and QA: CORAS supports QA by intensive communication between analysts and clients. The feedback from analysts and clients leads to improvements in the quality of the artifacts produced. The CORAS security risk modelling language [89] is equipped with a structured semantics, which explains step-by-step how a graphical CORAS model can be interpreted and how it can be translated into an English text.

(4) Relation to conceptual framework: The notions security goal, assumption, risk, threat, attack, and vulnerability match the notions of our CF. Most other notions given in our CF are not considered by CORAS.

8.2 Tropos goal-risk framework

(1) Description: Asnar et al. [90] propose the Tropos Goal-Risk Framework, an extension of earlier work [91], to assess risk based on *trust relations* among actors. More precisely, the extension comprises the introduction of the notion trust as a "subjective probability that defines the expectation of an actor about profitable behavior of another actor" [90]. Trust, combined with the concept of *delegation* of the fulfillment of a goal, enables the modeling of responsibility transfer from one actor to another. The authors introduce a three-layer model that comprises the *goal*, *event*, and *treatment* layers as an extension of the original Tropos goal model. Goals are AND/OR decomposed and related to external events that can negatively influence their satisfaction. Treatments are introduced to mitigate the effects of such events. The authors propose qualitative risk reasoning techniques to support the analyst in evaluating and choosing among different possible subgoal trees. Additionally, the method is tool supported by the GR-Tool[10]. The method is described by a set of algorithms written in pseudo code notation.

(2) Scope: The proposed Goal-Risk Framework by Asnar et al. [90] comprises all software development steps covered by Tropos. It covers security requirements elicitation and analysis based on risk analysis.

(3) Validation and QA: QA is included in the approach by integrating risk analysis techniques into security requirements engineering. Risk analysis is used to evaluate alternative goals and to assess countermeasures to mitigate risks.

(4) Relation to conceptual framework: Since Asnar et al. make use of the Tropos modeling framework, the notions security goal and security requirement correspond to the equally named notions of our conceptual framework. The notion security requirement is also used to refer to a specification. Furthermore, the notions risk and threat (also called event) match the equally named notions of our conceptual framework. The work by Asnar et al. does not consider the notions domain knowledge, asset, and vulnerability given in the conceptual framework.

8.3 Model-based information system security risk management (ISSRM)

(1) Description: Mayer et al. [66] propose a security requirements engineering process that consists of the following four steps: context analysis and asset identification, security goal determination, refinement of these goals to security requirements, and countermeasures selection. Both

[10] http://www.troposproject.org/tools/grtool/

of the latter two steps are based on a risk analysis approach named model-based ISSRM.

Thereby, Mayer et al. propose to make use of Yu's i* [57, 58] requirements engineering techniques, which can also be used to deal with security requirements [69].

(2) Scope: The proposed method by Mayer et al. comprises security requirements elicitation driven by a risk analysis method. It also supports analyzing security requirements through context and asset analysis.

(3) Validation and QA: The work by Mayer et al. does not describe special QA treatments. Since it makes use of the i* modeling framework, possibly QA procedures from this requirements engineering approach can be applied.

(4) Relation to conceptual framework: Mayer et al. make use of the i* modeling framework. Hence, the notions security goal and security requirement correspond to the equally named notions of our CF. The notion security requirement is also used to refer to a specification. Furthermore, the notions assumption, risk, threat, attack, and vulnerability match the equally named notions of our CF. The work by Mayer et al. does not consider the remaining notions given in our CF.

9 Common criteria-based approaches

9.1 Common Criteria (CC)

(1) Description: The Common Criteria are an international standard to achieve comparability between the results of independent security evaluations of IT products (machines). Such a machine, which may consist of hardware and software, is called Target of Evaluation (TOE). The name *Common Criteria* is an abbreviation for the Common Criteria for Information Technology Security Evaluation [1]. The current version is 3.1, dating from September 2006. The CC are also known as ISO/IEC 15408. The CC are compiled by a consortium of governmental organizations. Contributing countries include Australia/New Zealand, Canada, France, Germany, Japan, Netherlands, Spain, United Kingdom, and the United States.

With respect to notation, the CC use natural language for all of its concepts, as well as for the TOE and its properties. However, some concepts such as Security Functional Requirements (SFR) are formulated using a standardized language for enhanced exactness and comparability.

The CC standards present a General Model of some basic concepts (see Fig. 12, cf. [1]) that are drawn from traditional security engineering concepts. This model reflects the main scope and origin of the CC (evaluation of specific IT security products). Without adaptation and modification, however, it is less suited to be used as a single common ground for security requirements engineering (as proposed by [34]). Further, the General Model does not consider multilateral security.

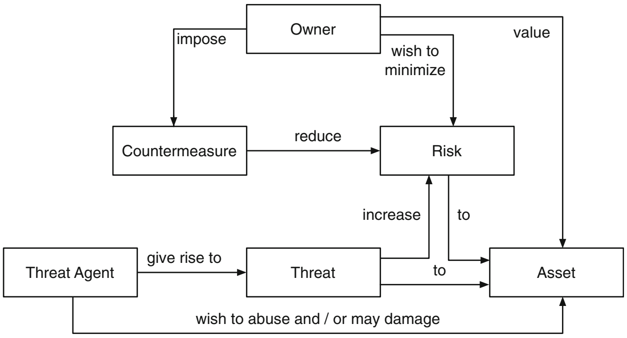


Fig. 12 Common Criteria—general model

The main concepts in Fig. 12 are defined as follows. Assets are defined as entities that the *owner of the TOE places value upon*; protecting assets is the responsibility of the TOE owner. Other stakeholders and their security goals are not considered here, though in further documents on SFR (see below) the protection of user data is mentioned. Threat agents seek to abuse assets in a manner contrary to the goals (interests) of the owner, leading to potential reduction of the asset value for the owner.

Threats include the loss of asset confidentiality, integrity or availability—but these threatened security goals are not made explicit in the CC model. This is a drawback for requirements engineering and gives also rise to CC-internal inconsistencies between asset definition and asset examples (e.g., in [1], Part 1, p. 34).

Threats imply risk to the assets, based on the likelihood and the impact of the threat being realized. Countermeasures are imposed by the owner to reduce the risks. These countermeasures comprise IT countermeasures (e.g., firewalls) and non-IT countermeasures (e.g., guards, procedures).

Three main phases can be identified in a CC process (cf. Fig. 13). Potential TOE owners infer from their security goals (called security needs in the CC) the need for specific types of security machines, which are types of TOE. These are refined into documents called Protection Profiles (PP). On the other hand, we have TOE developers or vendors who claim that their specific machine conforms to abstract PP. They publish their claims in documents called Security Targets (ST). In between, there is the actual CC evaluation process that checks these claims and can describe the confidence into its evaluation results according to standard assurance levels.

The specific concepts of the CC used in this evaluation process are defined as follows. The Target of Evaluation (TOE) is a *set of software, firmware, and/or hardware possibly accompanied by guidance*. For requirements

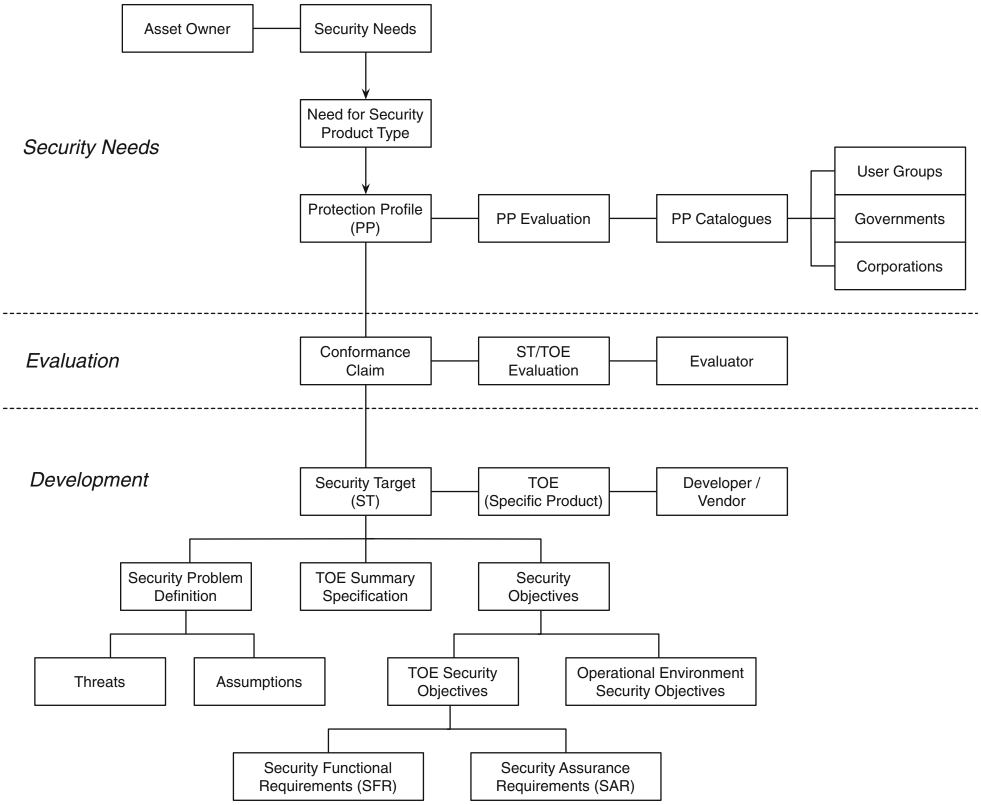


Fig. 13 Common Criteria—specific concepts

engineering purposes, it can be considered as the machine, or as part of a larger machine that is responsible for security functionality.

A Protection Profile (PP) is an *implementation-independent* statement of security needs for a TOE type. A PP describes a TOE *type* (e.g. firewalls). A PP is described as a *security specification on a relatively high level of abstraction* and should not contain detailed protocol specifications, or concrete descriptions of algorithms or operations. There are attempts to extend the PP concept to very large systems using system-level protection profiles (SLPP). For challenges involved in this approach, cf. [92].

In contrast to a PP, the Security Target (ST) describes a *specific* TOE (see Fig. 14). It is an *implementation-dependent* statement of security needs for a specific identified TOE. PP and ST have nearly the same structure. They contain an introduction, which includes descriptions of the TOE on different levels of abstraction. The Conformance Claim shows whether and to which PP the ST claims conformance. The Security Problem Definition includes the threats to be countered, the Organisational Security Policies (OSP) to be enforced, and assumptions to be fulfilled by the combination of the TOE and its operational environment.

Further, the ST and PP contain Security Objectives for the TOE and for the operational environment to solve the security problem at hand. These are refined into Security Requirements (in terms of the CC, that is functional specifications for a security machine) that provide a translation of the security objectives for the TOE into Security Functional Requirements (SFR) and that use a

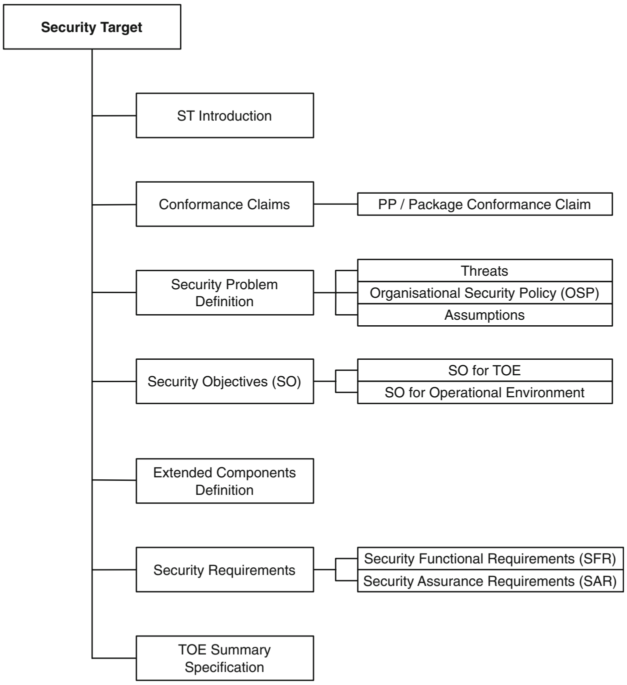

Security Target
ST Introduction
Conformance Claims
PP / Package Conformance Claim
Security Problem Definition
Threats
Organisational Security Policy (OSP)
Assumptions
Security Objectives (SO)
SO for TOE
SO for Operational Environment
Extended Components Definition
Security Requirements
Security Functional Requirements (SFR)
Security Assurance Requirements (SAR)
TOE Summary Specification

Fig. 14 Common Criteria—security target (ST)

standardized language. In addition, Security Assurance Requirements (SAR) for the evaluation process are stated. Finally, the ST (but not the PP) contains a TOE Summary Specification that indicates how the SFRs are implemented in the TOE.

The CC make use of patterns. The SFR used in the CC are textual patterns for specifications. SFR can be combined into named sets in the CC, for example, packages, families, and classes. One explicit goal of the CC is to make these patterns available in public catalogs for reuse. In a different sense, the whole PP is a pattern for a type of (security) machine and its specification.

(2) Scope: The CC standard itself does not present an actual requirements engineering method.

However, it is an important standard in the field of security engineering, and establishes its own body of security engineering concepts and standardized language. It influences security requirements engineering methods such as SREP [93] (discussed in the next section).

The CC are mainly concerned with security functionality and its assurance.

The CC are not concerned with legal aspects, evaluation of cryptographic algorithms or administrative security measures. Assumptions on the operational environment are to be stated explicitly.

CC evaluations can comprise all three major high-level goals of information security (confidentiality, integrity, availability), as well as many subaspects, which are refined in high detail using the corresponding catalogs. Threats, risk, and countermeasures are an integral part of the method as well. These concepts are included in the General Model of the CC (cf. Fig. 12). Threats are an important part of a CC evaluation and are included in the corresponding documents, such as the Protection Profile and the Security Target. The actual evaluation methodology is described separately.

(3) Validation and QA: In general, it is the task of the (potential) TOE owners to check for the completeness of the requirements, as well as to check if their security needs are met by the security target. Large user groups, corporations, and governments can develop and evaluate Protection Profiles that can be incorporated into public PP catalogs and reused. This procedure should increase their soundness and relative completeness. In addition, dependencies between Security Functionality Requirements (SFR) are explicitly considered, which aids in achieving completeness. However, the CC standard does not provide a formal semantic model.

The actual evaluation method for the CC is not part of the main standard. But one of the main goals of the CC is to assign an assurance level to the result of evaluation procedures. Assurance in CC terms is defined as *grounds for confidence that a TOE meets the SFRs*. To express and compare this degree of confidence, assurance packages such as the Evaluation Assurance Levels (EAL 1-7) can be used, cf. [1] Part 3.

Security requirements in the CC are functional (or assurance) requirements; other functional or non-functional requirements are not considered if they are not "relevant" to the security functionality. However, no systematic approach is presented to identify this relevance. Requirements conflicts in general are not explicitly considered, and correspondingly there are no ideas or methods for negotiation presented in the CC.

(4) Relation to conceptual framework: The CC security needs correspond to security goals in our CF. The security objectives in the PP could be mapped not only to our security requirements, whereas the security objectives in the ST and especially the SFR correspond to specifications, but also to more detailed design and implementation properties. The CC assumptions and security objectives for the operational environment refer to domain knowledge in our CF.

9.2 Security requirements engineering process (SREP)

(1) Description: Mellado et al. [93, 94] present a *Security Requirements Engineering Process* (SREP). SREP is an iterative and incremental security requirements engineering process, which is based on the Unified Process [95] software life-cycle model with multiple phases. Further, SREP is asset-based, risk driven, and, following the Common Criteria [1] (CC) supports the reuse of security requirements, as well as the reuse of knowledge on assets, threats, and countermeasures.

In SREP, UML [44] use cases are used to model security objectives, and misuse cases (see Sect. 5.1) are used to elicit threats [96]. Furthermore, a template is suggested for ranking threats, attacks, and risks. Security objectives and threats are modeled with use case and misuse case diagrams, threat specification is done using template-based misuse cases, while security requirements are specified using template-based use cases.

Based on the CC, the authors propose a *Security Resources Repository (SRR)*, which stores all the reusable security elements. A metamodel of the SRR is shown in Fig. 15 where the darker objects identify the extension of SREP to the repository metamodel as suggested by [96].

SREP is applied by working through the following nine activities:

1. Agree on definitions: The development or security analysis teams agree on a set of security definitions, organizational security policies, and the security visions of the information system. These definitions are then built into the *Security Vision Document*, which lists the most important assets of the information system.

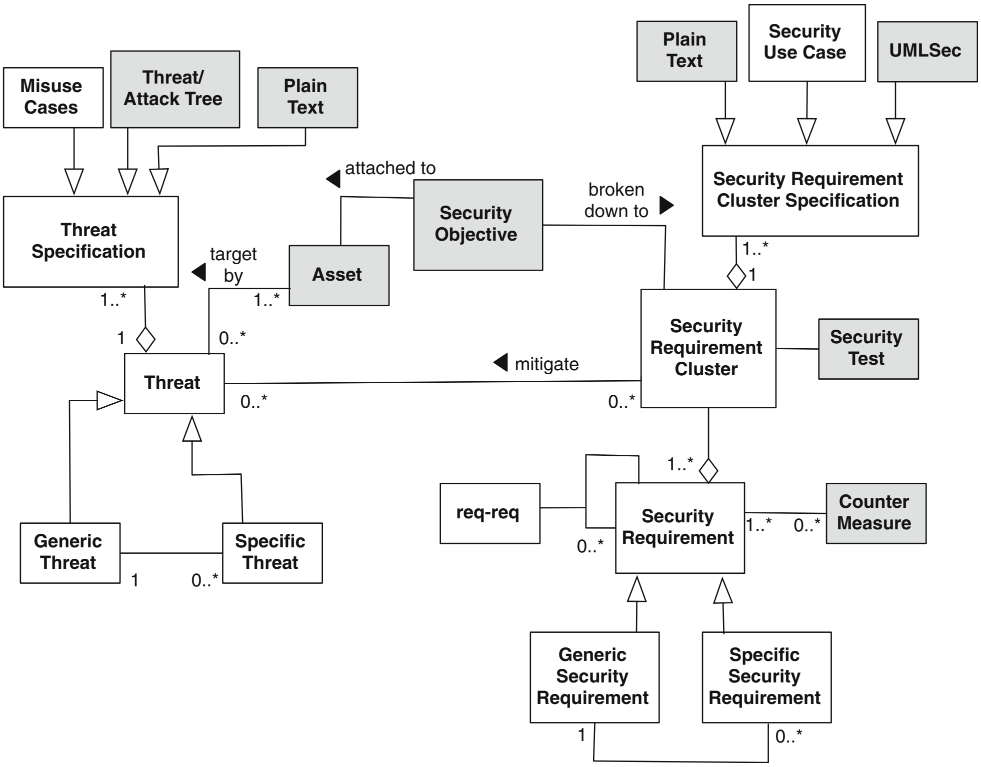


Fig. 15 SREP metamodel for the security resources repository (SRR)

2. Identify vulnerable and/or critical assets: An analysis of the functional requirements identifies important assets. Assets can be information, tangible assets (money, products), or intangible assets (reputation). The result is a more in-depth analysis of assets than in the Security Vision Document.
3. Identify security objectives and dependencies: If the type of assets identified in the previous activity can be found in the SRR, then their associated security objectives are retrieved. If not, the security objectives for each asset are determined, also taking into account organizational and legal restrictions. The list of security objectives should be refined in subsequent iterations by establishing dependencies between the security objectives. These are then compiled in the *Security Objectives Document*, using the CC assurance classes.
4. Identify threats and develop artifacts: If the assets can be found in the SRR, then it is possible to retrieve their associated threats from the repository. If not, use cases, misuse cases, and threat-attack trees can be applied. Further, public-domain sources and threat lists for the type of assets selected and following the CC assurance requirements for identifying potential vulnerabilities are utilized.
5. Risk assessment: The probability of each threat is determined, as well its potential impact and risk. The authors use tables (based on the method MAGERIT [97]), in which they quantify the impact and risk as well as the possible frequency of an attack. The results are captured in the *Risk Assessment Document*.
6. Elicit security requirements: Each security objective is analyzed for possible relevance and the threats it poses. This analysis is then used to identify the suitable security requirements that mitigate the threats at the necessary levels according to the risk assessment. Here, the interaction between security objectives, functional requirements, and threats are handled. The results are collected in the *Security Requirements Specification Document*.
7. Categorize and prioritize requirements: According to the risk, the security requirements are ranked.
8. Requirements inspection: The CC assurance requirements are used to validate the security requirements. If all security objectives are fulfilled, all security requirements are satisfied, and all the threats are countered, then the security problem is assumed to be solved. This result is then captured in the *Security Requirements Rationale Document*.
9. Repository improvement: The SRR can be extended with new elements. Part of this activity is the writing up of the *Security Target* (ST) document as it is defined in the CC.

None of the SREP activities has a formal foundation. The analysis of threats, attacks, and risks are indispensable to the method. They are used to elicit the security objectives and the security requirements.

(2) Scope: The method can be applied once the functional requirements analysis has been completed. Since SREP is based on the CC, it considers all three CIA goals.

(3) Validation and QA: SREP considers the completeness of the set of requirements in Activity 8. If all security requirements are satisfied, and all security objectives are achieved, and all the threats are countered, then the security problem is assumed to be solved. The consistency and completeness of requirements are improved by reusing (and iteratively refining) requirements in the SRR over several projects.

The interaction with functional requirements is considered in Activity 6. Other non-functional requirements are mentioned as important, but are not picked up in the method description. Conflicts are mentioned in the meta-model [93]. But, although the authors address the importance of analyzing conflicts, they do not come back to the topic in any of their activities.

(4) Relation to conceptual framework: The terminology of SREP is mapped onto our conceptual framework in the following manner. Security objectives stand for both the security goals and security requirements in the CF. The agreement of stakeholders is mentioned in the "requirements inspection" activity. Otherwise, stakeholder views are not part of the method. It is assumed that the elaboration of functional goals are completed. Other non-functional goals are not a focus of the method. The specification refers to both system requirements and the specification in the CF. Further, the specification is supposed to come close to a specification of the design of the security mechanisms to be implemented in the machine.

The analysis is made at the level of use cases, and hence circumstances or similar abstractions are not used. The authors perform a much more detailed taxonomy of assets and a continuous analysis of the interplay between risks, threats, and countermeasures. Preconditions in the use cases can be interpreted in some cases as environmental assumptions.

10 Comparison of security requirements engineering methods

Tables 5, 6, and 7 present mappings of main concepts of our CF to notions used by the previously presented SRE methods. Here, the notion security requirement (CF) for simplicity encompasses both the levels of security requirement and security system requirement in the abstraction hierarchy of security properties (Fig. 2), without referring to the state of reconciliation in detail.

A table entry labeled with ⊇ means that the notion defined in the considered approach is used in a narrower sense than the notion defined by our CF. An empty entry

Table 5 Mapping of notions used in SRE methods to our conceptual framework

Method	Notions of our conceptual framework		
	Security goal (CF)	Security requirement (CF)	Specification (CF)
MSRA	∼	∼	∼
SQUARE	∼	System-level req.	Software-level req.
Misuse cases	∼	–	Security req.
SecureUML	–	–	Security req.
UMLsec	–	–	Security req.
KAOS + anti-models	∼	∼	∼
Secure Tropos (Mouratidis et al.)	Softgoal, security constraint/feature, Secure entity	cf. Security goal	cf. Security goal
Secure i*	Softgoal / ∼	cf. Security goal	cf. Security goal
Secure Tropos (Massacci et al.)	Softgoal, security constraint/property	cf. Security goal	cf. Security goal
GBRAM	∼	∼	∼
Abuse frames	Security objective	Negated anti-req.	Security req.
SEPP	–	∼	∼
SREF	∼	∼	∼
CORAS	∼	–	–
Tropos goal-risk FW	Softgoal / ∼	cf. Security goal	cf. Security goal
Model-based ISSRM	Softgoal / ∼	cf. Security goal	cf. Security goal
CC	Security need	Security objective (PP)	Sec. objective (ST), SFR
SREP	Security objective	∼	∼

Author's personal copy

Table 6 Mapping of notions used in SRE methods to our conceptual framework

Method	Notions of our conceptual framework		
	Stakeholder (CF)	Domain knowledge (CF)	Asset (CF)
MSRA	∼	∼	Information
SQUARE	⊇ Client	–	–
Misuse cases	⊇ Actor	–	∼
SecureUML	⊇ User	–	–
UMLsec	⊇ Actor	Assumption	–
KAOS + anti-models	⊇ Agent	Domain properties, Expectation	⊇ object
Secure Tropos (Mouratidis et al.)	⊇ Actor	–	–
Secure i^*	⊇ Actor	–	∼
Secure Tropos (Massacci et al.)	⊇ Actor	–	–
GBRAM	∼	–	Asset/information
Abuse frames	⊇ Biddable domain	–	∼
SEPP	⊇ Biddable domain	Fact, assumption	⊇ Lexical domain, Phenomenon
SREF	⊇ Biddable domain	⊇ Fact, trust assumption	∼
CORAS	–	Assumption	∼
Tropos goal-risk FW	⊇ Actor	–	–
Model-based ISSRM	⊇ Actor	Context	∼
CC	TOE owner (general model), user (SFR)	Assumption, oper. env. Security objective	∼
SREP	–		∼

Table 7 Mapping of notions used in SRE methods to our conceptual framework

Method	Notions of our conceptual framework		
	Threat (CF)	Vulnerability (CF)	Risk (CF)
MSRA	–	–	–
SQUARE	∼	–	∼
Misuse cases	∼	–	∼
SecureUML	–	–	–
UMLsec	∼	∼	∼
KAOS + anti-models	∼	∼	–
Secure Tropos (Mouratidis et al.)	∼	∼	∼
Secure i^*	∼	∼	–
Secure Tropos (Massacci et al.)	–	–	–
GBRAM	∼	∼	∼
Abuse frames	∼	∼	–
SEPP	–	–	–
SREF	∼	–	∼
CORAS	∼	∼	∼
Tropos goal-risk FW	event / ∼	–	∼
Model-based ISSRM	∼	∼	∼
CC	∼	∼	∼
SREP	∼	∼	∼

means that a comparison is not possible because the considered notion is not defined in detail in this method. An entry labeled with—indicates that there is no comparable notion in the considered approach. Entries labeled with ∼ denote that the notion used in this approach matches the notion defined by our CF.

Author's personal copy

Table 8 Comparison of security requirements engineering methods

Method	Criteria									
	Considers:		Stakeholder views?	Multi-lateral?	Oriented toward		Includes		QA?	Formality?
	CIA	Other reqs.			System	Machine	Threats	Risks		
MSRA	×	×	×	×	×					
SQUARE	×	×	×	×	×	×	×	×	×	
Misuse cases	×	×				×	×	×		
SecureUML						×				×
UMLsec	×	×				×	×			×
KAOS + anti-models	×	×	×	×	×	×	×		×	×
Secure Tropos (Mouratidis et al.)	×	×	×	×	×	×			×	×
Secure i*	×	×	×	×	×	×	×		×	
Secure Tropos (Massacci et al.)	×	×	×	×	×	×			×	×
GBRAM					×	×	×	×	×	
Abuse frames					×	×	×			
SEPP		×			×	×			×	×
SREF	×	×			×	×	×		×	×
CORAS					×	×	×	×	×	
Tropos goal-risk FW	×	×	×		×	×	×	×	×	
Model-based ISSRM	×	×	×		×	×	×	×		
CC	×				×	×	×	×	×	
SREP	×				×	×	×	×	×	

Table 8 summarizes the presentation of the methods in the preceding sections. The criteria in the table correspond to the central concepts of the framework presented in Sect. 2. An entry in the table indicates that the authors of a method explicitly consider a criteria. However, the free cells of the table do not imply that a method could not cover a criteria.

Most methods consider the complete CIA triad. Some of them also address other requirements than security requirements.

Although it is generally accepted that a unilateral view to security is outdated and the views of different stakeholders must be taken into account in SRE, only MSRA, SQUARE, KAOS and the Secure Tropos variants *explicitly* address this issue. This does not mean that it is impossible to consider the views of different stakeholders using the other methods. However, they do not capture this issue in their various activities.

Multilateral security stresses the fact that stakeholders do not only have justified different security concerns, but that these concerns also are often inherently contradictory, and a compromise must be established between them in a way that incorporates all stakeholders, explicitly. MSRA is the only method that proposes steps to address this issue.

There is no clear-cut distinction between methods addressing mostly security goals and requirements, and methods that are more oriented toward the machine and its specification. Only the UML-based methods do not take environmental issues into account, whereas only MSRA does not derive a specification of the machine.

All methods but MSRA and SecureUML consider threats. The concept of a counter-stakeholder in MSRA cannot be considered a threat agent, because it does not imply that the counter-stakeholder will threaten the system. SecureUML is concerned with access control only. Therefore, general threat analysis is out of the scope of this method.

Risk analysis is often considered an important aspect of security requirements engineering. Hence, the risk-based approaches, CC, SREP, and Misuse Cases put it into the center of attention. SQUARE devotes two steps to risk analysis. To some extent, GBRAM also considers risks.

The majority of the methods provide means for quality assurance.

Finally, while several methods cover most of the criteria mentioned in Table 8, the UML-based approaches and Abuse Frames have a narrower scope of application.

Author's personal copy

11 Conclusion and perspectives

This article contributes to security requirements engineering in two major aspects: first, it introduces a conceptual framework for security requirements engineering that relates the central concepts used in this field; second, it maps the diverse terminologies of different methods to that framework, facilitating access to those methods and their comparison.

There, apparently, is no established terminology for the field of security requirements engineering. The literature often uses the same notions, such as "requirement" or "asset", for different, though related, concepts. The conceptual framework of Sect. 2 distinguishes the different concepts such as security goals, requirements, specifications, and security properties, and thus provides a consistent terminology. Mapping the terminology of a particular method to the conceptual framework allows to assess the scope of the method—and, therefore, also its usefulness for a given purpose.

We consider a common case study to compare these methods as a very desirable effort for the future. A common illustrative example could help practitioners and academia to select a method that fits their application area best.

Moreover, the integration of the different SRE methods comprises a challenging, but worthwhile improvement for future research directions. In fact, methods such as the Tropos variants, KAOS, i*, and SEPP have strong focus on requirements engineering. Still, the artifacts produced during this development phase involve the difficulty of how to use them as a basis for the design phase. Therefore, the methods focusing on security requirements engineering should be combined with those that focus on the design of secure software and systems, such as UMLsec and SecureUML.

Also, the key features of the rather new developments such as the risk-driven approach model-based ISSRM, and the multilateral methods MSRA and SQUARE, should be integrated into the more sophisticated methods such as the Tropos variants and KAOS. A first attempt in this direction constitutes the Tropos Goal-Risk Framework [90].

Not least, the consideration of further quality requirements besides security, adaption to continuous changes in technology and business demands, and methods for requirement conflict resolution are important directions for future work.

Acknowledgments We thank the anonymous reviewers for their helpful comments and suggestions.

References

1. Common Criteria for Information Technology Security Evaluation, Version 3.1. (2006) [Online]. Available: http://www.commoncriteriaportal.org/public/expert/
2. Bishop M (2003) Computer security. Addison-Wesley, New York
3. Viega J, McGraw G (2001) Building secure software: how to avoid security problems the right way. Addison-Wesley, New York
4. Eckert C (2004) IT-Sicherheit, 3rd edn. Oldenbourg-Verlag, München
5. Firesmith DG (2003) Common concepts underlying safety, security, and survivability engineering. Carnegie Melon University. Technical report SEI-2003-TN-033
6. Rupp C, SOPHIST GROUP (2003) Requirements-engineering und -management, 3rd edn. Carl Hanser Verlag
7. Rannenberg K, Pfitzmann A, Müller G (1999) IT security and multilateral security. In: Müller G, Rannenberg K (eds) Multilateral security in communications—technology, infrastructure. Economy Addison-Wesley, pp 21–29
8. Zave P, Jackson M (1997) Four dark corners of requirements engineering. ACM Trans Softw Eng Methodol 6(1):1–30
9. Fricker S, Gorschek T, Glinz M (2008) Goal-oriented requirements communication in new product development. In: Proceedings of the international workshop on software product management. IEEE Computer Society, Los Alamitos, pp 27–34
10. Liu L, Yu E (2001) From requirements to architectural design using goals and scenarios. In: Proceedings of the international workshop from software requirements to architectures (STRAW). Toronto
11. Antòn AI, Earp JB (2000) Strategies for developing policies and requirements for secure electronic commerce systems. Department of Computer Science, North Carolina State University. Technical report TR-2000-09. [Online]. Available: citeseer.ist.psu.edu/anton00strategies.html
12. Mylopoulos J, Chung L, Nixon B (1992) Representing and using non-functional requirements: a process-oriented approach. IEEE Transactions on Software Engineering pp 483–497
13. Sommerville I (2007) Software Engineering, 8th edn. Addison Wesley, New York
14. Glinz M (2007) On non-functional requirements. In: Proceedings of 15th IEEE international requirements engineering conference (RE '07), pp 21–26
15. Jureta I, Mylopoulos J, Faulkner S (2008) Revisiting the core ontology and problem in requirements engineering. In: Proceedings of 16th IEEE international requirements engineering conference (RE '08), pp 71–80
16. Information technology—security techniques—code of practice for information security management (ISO/IEC FDIS 17799:2005) (2005) International Organization for Standardization
17. Information technology—security techniques—management of information and communications technology security—part 1: Concepts and models for information and communications technology security management (ISO/IEC 13335-1:2004)(2004) International Organization for Standardization
18. NIST SP 800-26: Security Self-Assessment Guide for Information Technology Systems (2001) National institute of standards and technology
19. Berry DM, Lawrence B (1998) Guest editors' introduction: requirements engineering. IEEE Softw 15(2):26–29
20. Robinson WN, Pawlowski SD, Volkov V (2003) Requirements interaction management. ACM Comput Surv 35(2):132–190
21. Finkelstein A, Baggay D, Hunter A, Kramer J, Nuseibeh B (1994) Inconsistency handling in multi-perspective specifications. IEEE Trans Softw Eng (20):569–578
22. Easterbrook S, Nuseibeh B (1996) Using viewpoints for inconsistency management. Softw Eng J 31–43
23. Kotonya G, Sommerville I (1996) Requirements engineering with viewpoints. BCS/IEE Softw Eng J 11(1):5–18
24. Giorgini P, Massacci F, Mylopoulos J, Zannone N (2006) Detecting conflicts of interest. In: Proceedings 14th IEEE

international requirements engineering conference (RE '06). IEEE Computer Society, pp 308–311
25. van Lamsweerde A, Darimont R, Massonet P (1998) Managing conflicts in goal-driven requirements engineering. IEEE Trans Softw Eng 24
26. Jackson M, Zave P (1995) Deriving specifications from requirements: an example. In: Proceedings 17th international conference on software engineering. ACM Press, Seattle, pp 15–24
27. Haley B, Laney C, Moffett D, Nuseibeh B (2006) Using trust assumptions with security requirements. Requir Eng 11(2):138–151
28. Haley CB, Laney R, Moffett J, Nuseibeh B (2008) Security requirements engineering: a framework for representation and analysis. IEEE Trans Softw Eng 34(1):133–153
29. Santen T (2006) Stepwise development of secure systems. In Górski J (ed) International conference on computer safety, reliability and security (SAFECOMP), ser. LNCS 4166. Springer, pp 142–155
30. Moffett JD, Haley CB, Nuseibeh B (2004) Core security requirements artifacts. The Open University, UK (technical report)
31. Breaux TD, Antòn A (2005) Analyzing goal semantics for rights, permissions, and obligations. In: Requirements engineering, pp 177–188
32. Mayer N (2009) Model-based management of information system security risk. Ph.D. dissertation, University of Namur [Online]. Available: http://www.nmayer.eu/publis/Thesis_Mayer_2.0.pdf
33. Mayer N, Heymans P, Matulevičius R (2007) Design of a modelling language for information system security risk management. In: 1st International conference on research challenges in information science (RCIS 2007)
34. Mellado D, Fernandez-Medina E, Piattini M (2006) A comparison of the Common Criteria with proposals of information systems security requirements. In: ARES '06: proceedings of the first international conference on availability, reliability and security (ARES'06). IEEE Computer Society, Washington, DC, pp 654–661
35. Kalloniatis C, Kavakli E, Gritzalis S (2004) Security requirements engineering for e-government applications: analysis of current frameworks. Springer, Berlin
36. Tøndel I, Jaatun M, Meland P (2008) Security requirements for the rest of us: asurvey. Softw IEEE 25(1):20–27
37. van Lamsweerde A (2007) Engineering requirements for system reliability and security. In: Broy JGM, Hoare C (eds) Software system reliability and security, ser. NATO security through science series-D: information and communication security, vol 9. IOS Press, pp 196–238
38. Gürses S, Santen T (2006) Contextualizing security goals—a method for multilateral security requirements elicitation. In: Dittmann J (ed) Proceedings of Sicherheit 2006—Schutz und Zuverlässigkeit, ser. Lecture notes in Informatics. Gesellschaft für Informatik, pp 42–53
39. Gürses S, Berendt B, Santen T (2006) Multilateral security requirements analysis for preserving privacy in ubiquitous environments. In: Berendt B, Menasalvas E (eds) Proceedings of workshop on ubiquitous knowledge discovery for users (UKDU'06) [Online]. Available:http://www.vasarely.wiwi.hu-berlin.de/UKDU06/Proceedings/UKDU06-proceedings.pdf
40. Gürses S, Jahnke JH, Obry C, Onabajo A, Santen T, Price M (2005) Eliciting confidentiality requirements in practice. In: CASCON '05: Proceedings of the 2005 conference of the centre for advanced studies on collaborative research. IBM Press, pp 101–116
41. Onabajo A, Weber-Jahnke J (2008) Stratified modeling and analysis of confidentiality requirements. In: 41st Annual Hawaii international conference on system sciences
42. Mead N, Hough E, Stehney T (2005) Security quality requirements engineering (SQUARE) methodology. Carnegie Mellon Software Engineering Institute, Technical report CMU/SEI-2005-TR-009
43. Mead N, Viswanathan V, Padmanabhan D, Raveendran A (2008) Incorporating security quality requirements engineering (SQUARE) into standard life-cycle models. Carnegie Mellon Software Engineering Institute. Technical report CMU/SEI-2008-TN-006
44. UML Revision Task Force (2006) OMG unified modeling language: superstructure. http://www.omg.org/docs/ptc/06-04-02.pdf
45. Sindre G, Opdahl AL (2001) Capturing security requirements by misuse cases. In: Proceedings of the 14th Norwegian informatics conference (NIK'2001)
46. Sindre G (2007) Mal-activity diagrams for capturing attacks on business processes. In: Sawyer P, Paech B, Heymanns P (eds) Proceedings of REFSQ 2007, ser. LNCS 4542. Springer, pp 355–366
47. Lodderstedt T, Basin DA, Doser J (2002) SecureUML: a UML-based modeling language for model-driven security. In: Proceedings of the 5th international conference on the unified modeling language (UML'02). Springer, London, pp 426–441
48. UML Revision Task Force (2006) OMG object constraint language: reference. http://www.omg.org/docs/formal/06-05-01.pdf
49. Jürjens J (2003) Secure systems development with UML. Springer, New York
50. Bertrand P, Darimont R, Delor E, Massonet P, van Lamsweerde A (1998) GRAIL/KAOS: an environment for goal drivent requirements engineering. In: ICSE'98—20th international conference on software engineering
51. Dardenne A, van Lamsweerde A, Fickas S (1993) Goal-directed requirements acquisition. Sci Comput Program 20(1–2):3–50
52. van Lamsweerde A (2004) Elaborating security requirements by construction of intentional anti-models. ICSE pp. 148–157
53. Bresciani P, Perini A, Giorgini P, Giunchiglia F, Mylopoulos J (2004) Tropos: an agent-oriented software development methodology. Auton Agent Multi Agent Syst 8(3):203–236
54. Giorgini P, Susi A, Perini A, Mylopoulos J (2005) The tropos metamodel and its use. Inf J 29:401–408
55. Fuxman A, Liu L, Mylopoulos J, Pistore M, Roveri M, Traverso P (2004) Specifying and analyzing early requirements in tropos. Requir Eng J 9(2):132–150
56. Yu ES-K (1996) Modelling strategic relationships for process reengineering. Ph.D. dissertation, University of Toronto, Toronto
57. Yu ESK (1997) Towards modeling and reasoning support for early-phase requirements engineering. In: RE '97: proceedings of the 3rd IEEE international symposium on requirements engineering. IEEE Computer Society, Washington, DC, p 226
58. Yu ESK, Liu L (2001) Modelling trust for system design using the i^* strategic actors framework. In: Proceedings of the workshop on deception, fraud, and trust in agent societies held during the autonomous agents conference. Springer, London, pp 175–194
59. Giorgini P, Mouratidis H, Zannone N (2007) Modelling security and trust with secure tropos. In: Integrating security and software engineering: advances and future vision. IDEA
60. Mouratidis H, Giorgini P (2007) Secure tropos: a security-oriented extension of the tropos methodology. Int J Softw Eng Knowl Eng 17(2):285–309
61. Mouratidis H, Giorgini P (2004) Enhancing secure tropos to effectively deal with security requirements in the development of multiagent systems. In: Proceedings of the 1st international workshop on safety and security in multiagent systems, SASEMAS
62. Mouratidis H, Giorgini P (2005) Secure tropos: dealing effectively with security requirements in the development of multi-agent systems. In: Proceedings of the 2nd international workshop on safety and security in multi-agent systems, SASEMAS, ser. Computers & Security, vol 24, no.8. Elsevier, pp 614–617

63. Massacci F, Mylopoulos J, Zannone N (2007) Ontologies for business interaction. Information science reference, ch. An ontology for secure socio-technical systems pp 188–207
64. Elahi G, Yu E (2007) A goal oriented approach for modeling and analyzing security trade-offs. University of Toronto, Department of Computer Science. Technical report
65. Matulevičius R, Mayer N, Mouratidis H, Dubois E, Heymans P, Genon N (2008) Adapting secure tropos for security risk management in the early phases of information systems development. In: CAiSE '08: proceedings of the 20th international conference on advanced information systems engineering. Springer, Berlin, pp 541–555
66. Mayer N, Rifaut A, Dubois E (2005) Towards a risk-based security requirements engineering framework. In: Proceedings of the 11th international workshop on requirements engineering: foundation for software quality (REFSQ'05), in conjunction with the 17th conference on advanced information systems engineering (CAiSE'05)
67. Bauer B, Müller JP, Odell J (2001) Agent UML: a formalism for specifying multiagent software systems. Int J Softw Eng Knowl Eng 11(3):207–230
68. Giorgini P, Manson G, Mouratidis H (2004) Using security attack scenarios to analyse security during information systems design. In: The 6th international conference on enterprise information systems. Porto
69. Liu L, Yu E, Mylopoulos J (2003) Security and privacy requirements analysis within a social setting. In: Proceedings of 11th IEEE requirements engineering conference. IEEE Press, pp 151–161
70. Abiteboul S, Hull R, Vianu V (1995) Foundations of databases. Addison-Wesley, New York
71. Giorgini P, Massacci F, Mylopoulos J, Zannone N (2005) St-tool: a case tool for security requirements engineering. In: RE-05. IEEEP, pp 451–452
72. Massacci F, Zannone N (2006) Detecting conflicts between functional and security requirements with secure tropos: John rusnak and the allied irish bank
73. Leone N, Pfeifer G, Faber W, Eiter T, Gottlob G, Perri S, Scarcello F (2006) The DLV system for knowledge representation and reasoning. ACM Trans Comput Logic 7(3):499–562
74. He Q, Antòn AI (2003) A framework for modeling privacy requirements in role engineering. In: International workshop on requirements engineering for software quality (REFSQ 2003)
75. CERIAS Technical Report (1999) Policy framework for interpreting risk in ecommerce security
76. Hauser J, Clausing D (1988) The house of quality. Harv Bus Rev 32(5)
77. Jackson M (2001) Problem frames. Analyzing and structuring software development problems. Addison-Wesley, New York
78. Lin L, Nuseibeh B, Ince D, Jackson M (2004) Using abuse frames to bound the scope of security problems. In: Proceedings of 11th IEEE international requirements engineering conference (RE'04). pp 354–355
79. Hatebur D, Heisel M, Schmidt H (2006) Security engineering using problem frames. In: Müller G (ed) Proceedings of the international conference on emerging trends in information and communication security (ETRICS'06), ser. LNCS 3995. Springer, pp 238–253
80. Hatebur D, Heisel M, Schmidt H, (2007) A pattern system for security requirements engineering. In: Proceedings of the international conference on availability, reliability and security (AReS). IEEE Computer Society, pp 356–365
81. Hatebur D, Heisel M, Schmidt H (2007) A security engineering process based on patterns. In: Proceedings of the international workshop on secure systems methodologies using patterns (SPatterns). IEEE Computer Society, pp 734–738
82. Hatebur D, Heisel M, Schmidt H (2008) Analysis and component-based realization of security requirements. In: Proceedings of the international conference on availability, reliability and security (AReS). IEEE Computer Society, pp 195–203
83. Schmidt H (2009) Pattern-based confidentiality-preserving refinement. In: Engineering secure software and systems—first international symposium (ESSoS), ser. LNCS, vol 5429. Springer, Berlin, pp 43–59
84. Schmidt H, Wentzlaff I (2006) Preserving software quality characteristics from requirements analysis to architectural design. In: Proceedings of the European workshop on software architectures (EWSA), vol 4344/2006. Springer, Berlin, pp 189–203
85. Haley CB, Moffett JD, Laney R, Nuseibeh B (2006) A framework for security requirements engineering. In: SESS '06: proceedings of the 2006 international workshop on Software engineering for secure systems. ACM Press, New York, pp 35–42
86. Haley C, Laney R, Moffett J, Nuseibeh B (2004) Picking battles: the impact of trust assumptions on the elaboration of security requirements. In: Jensen CD, Poslad S, Dimitrakos T (eds) iTrust'04, pp 347–354
87. Haley CB, Moffett JD, Laney R, Nuseibeh B (2005) Arguing security: validating security requirements using structured argumentation. In: Proceedings of the 3rd symposium on requirements engineering for information security (SREIS'05). Paris
88. Braber F, Hogganvik I, Lund MS, Stølen K, and Vraalsen F (2007) Model-based security analysis in seven steps—a guided tour to the CORAS method. BT Technol J 25(1):101–117
89. Dahl HEI, Hogganvik I, Stølen K (2007) Structured semantics for the CORAS security risk modelling language. SINTEF information and communication technology Technical report STF07 A970
90. Asnar Y, Giorgini P, Massacci F, Zannone N (2007) From trust to dependability through risk analysis. In: Proceedings of the international conference on availability, reliability and security (AReS). IEEE Computer Society, pp 19–26
91. Asnar Y, Giorgini P, Mylopoulos J (2006) Risk modelling and reasoning in goal models. University of Trento. Technical report DIT-06-008
92. Keblawi F, Sullivan D (2006) Applying the common criteria in systems engineering. IEEE Secur Priv 4(2):50–55
93. Mellado D, Fernandez-Medina E, Piattini M (2006) Applying a security requirements engineering process. In: ESORICS'06
94. Mellado D, Fernander-Medina E, Piattini M (2006) A comparison of the common criteria with proposals of information systems security requirements. In: First international conference on availability, reliability, and security (ARES'06). pp 654–661
95. Booch G, Rumbaugh J, Jacobson I (1999) The Unified Software Development Process. Addison-Wesley, New York
96. Sindre G, Firesmith DG, Opdahl AL (2003) A reuse-based approach to determining security requirements. In: Ninth international workshop on requirements engineering (REFSQ'03). http://www.citeseer.ist.psu.edu/580371.html
97. MAP (2005) Metodologìa de anàlisis y gestiòn de riesgos de los sistemas de informaciòn (magerit-v 2)