Supporting Common Criteria Security Analysis with Problem Frames*

Kristian Beckers[1†], Maritta Heisel[1], and Denis Hatebur[2]
[1] *paluno*
University of Duisburg-Essen, Germany
{kristian.beckers, maritta.heisel}@uni-due.de
[2] *ITESYS*
Institute for Technical Systems GmbH, Germany
d.hatebur@itesys.de

Abstract

Security standards, e.g., the Common Criteria (ISO 15408), are applied by software vendors to establish a level of confidence that the security functionality of their products and their applied assurance measures are sufficient. To get a Common Criteria certification, a comprehensible set of documents is necessary, including a detailed threat analysis and security objective elicitation. We focus on improving the Common Criteria threat analysis and the derivation of security objectives in our work.

Our method is based upon an attacker model, which considers different attacker types, e.g., software attackers, that threaten only specific parts of a system. We provide tool support for checking the consistency and the completeness of the specified software systems using OCL expressions. For example, we check if all types of attackers have been considered for a specific domain, we check for all software domains that either a software attacker is considered or an assumption is documented that excludes software attackers, and we check if all threats are addressed by security objectives. Moreover, we can generate tables and texts from our UML models to satisfy the Common Criteria documentation demands. For instance, we can generate Common Criteria specific cross-table, which maps every security objective and assumption to a specific threat. The consistency checks are integrated in our structured method for threat analysis that considers the Common Criteria's (CC) demands for documentation of the system in its environment and the reasoning that all threats are discovered and addressed. With our support tool UML4PF (that extends a UML tool and contains e.g., a UML profile and an OCL validator), we support security reasoning, validation of models, and we are able to generate Common Criteria-compliant documentation using model-to-text transformations. Our threat analysis method can also be used for threat analysis without the common criteria, because it uses a specific part of the UML profile that can be adapted to other demands with little effort. For example, it could be adapted for other security standards like ISO 27001. We illustrate our approach with the development of a smart metering gateway system.

Keywords: common criteria, problem frames, security standards, document generation, security requirements engineering

Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications, volume: 5, number: 1, pp. 37-63

*This research was partially supported by the EU project Network of Excellence on Engineering Secure Future Internet Software Services and Systems (NESSoS, ICT-2009.1.4 Trustworthy ICT, Grant No. 256980) and the Ministry of Innovation, Science, Research and Technology of the German State of North Rhine-Westphalia and EFRE (Grant No. 300266902 and Grant No. 300267002).

†Corresponding author: Universität Duisburg-Essen, Fakultät für Ing.-wissenschaften, Abteilung INKO, Fachgebiet Software Engineering, 47048 Duisburg, Tel: +49-(0)2033791797

1 Introduction

Software vendors have to establish a level of confidence that the security functionality of their products and their applied assurance measures are sufficient. This can be done by a certification according Security standards, e.g., the Common Criteria (ISO 15408, CC). A certification requires a consistent and understandable set of documents. In this paper, we introduce a method to create documents necessary for a CC certification.

We reworked and integrated results from previous publications about the creation of Security Targets [1] and the problem-based CC compliant threat analysis [2]. We added the definition of *security objectives* being the next steps of our the method. Additionally, we extended and improved the model validation, the support tool and the capabilities of our document generation functionality.

We proposed to extend the problem frame method (see Sect. 2.2) to support the security reasoning and documentation demands of the CC (see Sect. 2.1). The CC demand several steps, such as a description of the software and hardware in its environment, asset identification, and threat analysis, but does not provide a description of when these steps are complete and all threats are elicited. A detailed description on how to conduct these steps is also not available. We provide a method that extends the UML4PF profile (see Sect. 2.2) with CC-specific terminology, e.g., the so-called *ToE* is the system containing hardware and software to be certified. In addition, we introduce an attacker classification and define specific kinds of Jackson's domains that these attackers threaten, e.g., a software attacker threatens only causal domains. Our method is supported by a modeling tool that contains OCL expressions [3], which support security reasoning by checking for completeness of the threat analysis. For example, it checks whatever all network connections are threatened by a network attacker. These expressions also check for consistency problems in the resulting model, e.g., if assets are also attackers.

Our method uses models and inserts all relevant information and texts in these models. We support the creation of security target (ST) and protection profile (PP) documents. We support the creation of all sections from the *introduction* section to the security objectives section (see Sect. 2.1). This requires a transformation from our UML models into texts and tables. We provide tool support that supports an automatic transformation from our models to tables and texts for ST and PP documents. We limit our work to security objectives, because these correspond to requirements (see our conceptual framework in [4]), and the problem frames method focuses on the requirements phase of software engineering. The support for the sections in ST and PP documents, e.g., security functional requirements that refer to software design are part of our future work. We validated our work by applying our method to an existing CC protection profile for a smart metering gateway [5].

2 Background

2.1 Common Criteria

The ISO/IEC 15408 - Common Criteria for Information Technology Security Evaluation is a security standard that can achieve comparability between the results of independent security evaluations of IT products (machines). These are so-called *Targets of Evaluation (TOEs)*. The Common Criteria are based upon a general security model. The model considers *ToE Owner*s that value their *Asset*s and wish to minimise *Risk* to these *Asset*s via imposing *Countermeasure*s. *Attacker*s wish to abuse *Asset*s and give rise to *Threat*s for *Asset*s. The *Threat*s increase the *Risk*s to *Asset*s.

Documentation of the security model is the basis for CC certification. The CC security model can be expressed in two different types of documents. The security needs of *ToE* owners are described in the so-called *Security Target (ST)*. An *ST* can be a refinement of a so-called *Protection Profile (PP)*. A *PP* states the security needs for an entire class of *ToE*s, e.g., client VPN application. A *PP* states the security requirements of *ToE* owners, and *ToE* developers or vendors publish their security claims in an *ST*.

The document structure of *ST* and *PP* is the same on the level of chapters. The first chapter is an *Introduction* that contains the description of the ToE and its environment. The chapter *Conformance Claims* describes to which *PP*s the *ST* or *PP* is compliant.

The chapter *Security Problem Definition* refines the external entities, e.g., stakeholders in the environment. In addition, the chapter lists all *Asset*s, *Assumption*s about the *ToE* and its environment as well as all *Attacker*s, the *Threat*s they cause to *Asset*s, and *Organizational Security Policies* of the *ToE*'s environment. The chapter *Security Objectives* contains the *Security Objectives* for the *ToE* and its *Operational Environment*. An example for an *Operational Environment* is the operating system the *ToE* uses.

2.2 Problem Frames

We use a requirements engineering method inspired by Jackson [6]. Requirements can only be guaranteed for a certain context. Therefore, it is important to describe the *environment*, because we build a system to improve something in the world. The environment in which the system to be built (called *machine*) will operate is represented by a *context diagram*.

We use the UML [7] notation with stereotypes defined in the UML profile UML4PF [8] to create a context diagram and domain knowledge diagrams. Stereotypes give a specific meaning to the elements of a UML diagram they are attached to, and they are represented by labels surrounded by double angle brackets. The class with the stereotype *machine* represents the thing to be developed (e.g., the software). The classes with some domain stereotype, e.g., *CausalDomain* or *BiddableDomain* represent *problem domains* that already exist in the application environment.

Domains are connected by interfaces consisting 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, as indicated by an exclamation mark.

Jackson distinguishes the domain types *CausalDomain*s that comply with some physical laws, *LexicalDomain*s that are data representations, and *BiddableDomain*s that are usually people. The stereotype <<causalDomain >> indicates that the corresponding domain is a *CausalDomain*, and the stereotype <<biddableDomain >> indicates that it is a *BiddableDomain*. In our formal meta-model of problem frames [9], *Domains* have *names* and *abbreviations*, which are used to define interfaces. Hence, the class *Domain* has the attributes *name* and *abbreviation* of type string. Requirements engineering by means of problem frames, proceeds as follows: It begins with a description of the desired functionality of the software to be built, the so-called *machine*. This description is refined into requirements and domain knowledge, which consists of facts and assumptions. An *osp* is an organizational security policy, which states rules that have to be followed when using the *ToE*. We use a *context diagram* and *domain knowledge diagram*s using our UML profile and tool support.

3 Supporting Common Criteria using Problem Frames

The ISO 15408 Standard - Common Criteria for Information Technology Security Evaluation (short CC) - [10] demands a detailed documentation of the software system that should be evaluated. This software system is the so-called *Target of Evaluation (ToE)* and consists of hard- and software. A *ToE* has to be described in detail, including its environment.

In this work, we focus on the threat analysis of the CC and on the description of the *ToE* in its environment, which is the input for this threat analysis. This considers *assets*, *attackers*[1], *threats*, *assumptions*, *security objectives*, and *security functional requirements* of the *ToE*. The challenge of any threat analysis is to achieve a complete coverage of all possible threats. Security requirements engineering (SRE) methods exist, which provide structured threat analysis on an abstraction of the system. However, these abstractions often only consider parts of the system-to-be [4], whereas for a CC-compliant threat analysis we require a complete model of the *ToE*.

Goal-based methods, e.g., SI* [11] and KAOS [12], investigate the goals and views of all stakeholders of the system. These methods model threats based upon structured goal models. Hence, they consider all goals and relevant software artifacts to these goals. However, they do not consider a complete view of the system-to-be. Other SRE methods follow similar steps, e.g., the asset-driven risk management method CORAS [13] identifies assets and determines threats to these assets. CORAS models the system-to-be in artifacts that have a relation to an asset and also does not represent the complete system-to-be. Therefore, we do not use any of these methods for our CC-compliant threat analysis.

The Problem Frames method [6] uses an abstraction of the system-to-be and models the environment of the system around it. Hence, this method is our choice for satisfying the CC's demand to model the *ToE* in its environment. The method models the *ToE* and its environment in domains with certain characteristics, and we propose a threat analysis that uses these characteristics to determine assets, possible attackers, and subsequent threats for these domains. We introduce a structured method that elicits attackers and threats for each domain. We also provide computer-aided support for consistency, document creation, and security reasoning for this method by using OCL queries on the problem frame models. Hence, we benefit from having a complete model of the system-to-be and its environment to conduct a threat analysis. Our method iterates over all domains of the system and reasons if these are threatened by an attacker. Hence, our method achieves completeness when all parts of the model are considered during security reasoning and the system model is complete.

4 UML Profile for problem-based and Common Criteria compliant Security Analysis

Our Common Criteria extension for the UML4PF profile is shown in Figs. 1 and 2. We split the profile into two figures in order to improve readability. All parts of the Common Criteria extension are marked in grey.

We contribute relations between problem frames and common criteria elements like *countermeasures*, which are now a kind of domain (see Fig. 1). The profile considers the *machine* to be evaluated, which is the *ToE*, and the *ToEOwner*, the person using the ToE. Assets are domains in Jackson's sense or part of a domain. We use an OCL expression to enforce this condition (see expression AE02CON in Tab. 1). *Asset*s have a *description* and a *need for protection* attribute.

[1]The CC uses the term *threat agent* for attacker. However, we use attacker as a synonym for threat agent in this work.

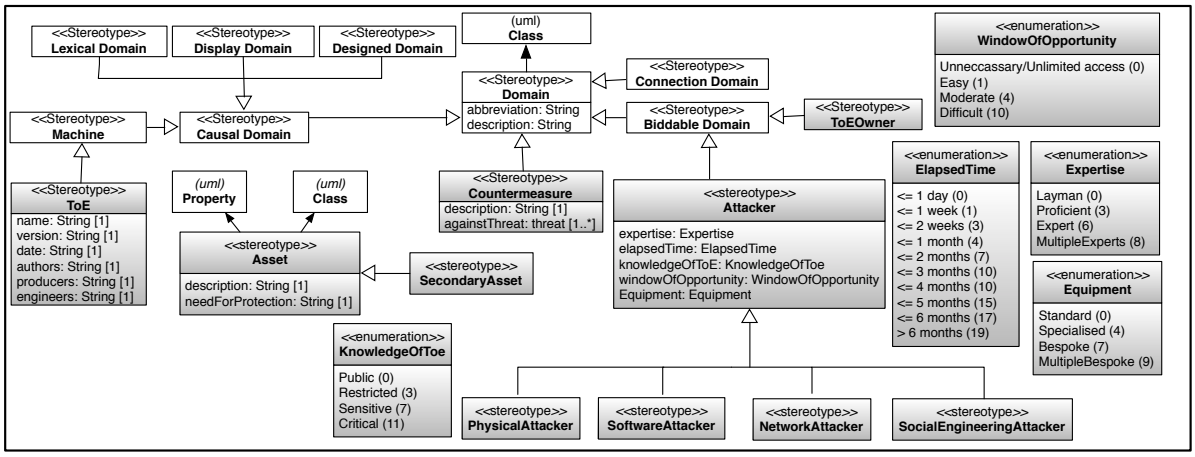

Figure 1: A Common Criteria extension of the UML4PF Profile (1/2)

The CC has also the concept of *SecondaryAssets*. Harm to *SecondaryAssets* do not cause a loss to the *ToEOwner* directly, but the harm can cause harm to an *Asset*. This in turn can cause a loss to a *ToEOwner*. *Threat*s can harm assets and have an abbreviation and a description. A *Risk* caused by threats has a likelihood and a consequence. An *osp* is an organizational security policy, which states rules that have to be followed when using the *ToE*. A *SpecificTerm* is a term that is not common knowledge is and defined for the ST/PP document. The specific term has a definition and a source of the definition.

Threats are caused by *Attacker*s that we classify into the following categories. *PhysicalAttacker*s threaten the physical elements of the system, e.g., hardware or buildings that host computers. *NetworkAttacker*s threaten *Network connections* or *ConnectionDomains* in our models. *SoftwareAttacker*s threaten the parts of the system that are software, e.g., the *ToE* or other *CausalDomains* that are a software. *SocialEngineeringAttacker*s threaten biddable domains, e.g., users of the system.

The UML profile also contains ≪enumerations≫ to represent the attributes required to describe an *Attacker* according to the CC. The attributes, e.g., *ElapsedTime* or *WindowOfOpportunity* have numeric values attached in brackets. These values are defined by the CC and are used to determine the EAL (evaluation assurance level) for a system. For example, an attacker with a combined score between 10 and 13 results in a recommendation to implement the security assurance classes *AVA_VAN.1* and *AVA_VAN.2*. This recommendation results in an *EAL* requirement of at least *EAL 2*. This classification of attackers is used in the CC during the evaluation of existing implementations. We propose to use it already during the requirements stage. Some information might not be available in the requirements phase, but the information that is already present can be included in the model and used for the security reasoning.

We extended the CC basic security model in order to distinguish different kinds of threats. *Threat*s can be further divided into *controlThreat*s and *observeThreat*s (see Fig. 2). *ControlThreat*s take control of a domain, while *observeThreat*s only observe information about the behavior of a domain. For example, an *observeThreat* is the eavesdropping of confidential information, whereas the manipulation of a key exchange is a *controlThreat*. The distinction between observe and control threats helps to determine the security objectives. For example, observe threats are likely to cause confidentiality problems. We propose a threat analysis during requirements engineering where the exact flow of information is only partially known. Hence,

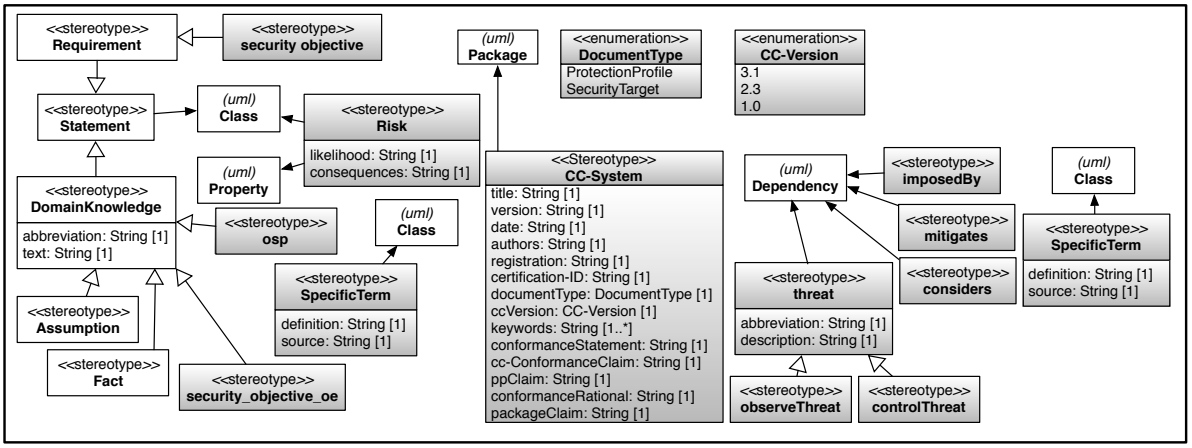

Figure 2: A Common Criteria extension of the UML4PF Profile (2/2)

we assume that informational assets can be reached by all domains or interfaces in the model. This is why we allow threats not only to be attached directly to assets, but to all domains or interfaces in our model. We explain how we model the ToE in its environment in Sect. 5.

Each *security objective mitigates* at least one threat and concern the ToE. *Security_objective_oe* state demands for the environment of the ToE. A security objective can also ≪referTo≫ a domain and ≪consider≫ an organizational security objective.

5 A Method for a Systematic Security Analysis and Documentation

Figure 3 shows our security analysis method, which we explain in the following.

1. Define scope To perform the security analysis systematically, we start with creating a context diagram that contains the scope of our analysis. The context diagram contains all domains (e.g., persons and technical systems) in the environment of the machine that are referred to by the functional requirements. For an example of a context diagram, see Fig. 4.

We defined several consistency checks for checking that a context diagram is correct. These are described in detail in [14] and are already part of UML4PF.

2. Asset identification For all domains in the context diagram, we check if the domain contains an *asset* or is an asset. Assets are documented in domain knowledge diagrams as classes with the stereotype ≪Asset≫ as introduced in the UML profile in Fig. 1. If the entire domain is an asset, we add the stereotype ≪Asset≫ to that class. In the case that an asset is only part of

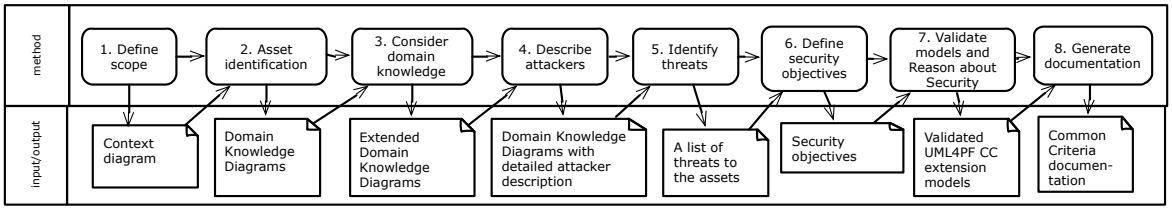

Figure 3: A Method for Common Criteria-Compliant Security Analysis

a domain, we use UML aggregation or composition relations between the asset and the domain it belongs to. We also identify secondary assets, which cause harm to other assets. We use OCL to check model consistency and completeness, which we also use for security reasoning. We state examples for each step of the method and how this step benefits from our OCL expressions. We provide an overview of OCL expressions for model consistency in Tab. 1. These expressions query the entire model, meaning context diagram and all domain knowledge diagrams. The table has a unique ID for each expression in the first column, the referenced class of the UML profile shown in Figs. 1 and 2 in the second column, and the consistency check the expression checks in the third column.

As an example, we discuss the OCL expression *AT01CON* in more detail, see *Listing 1*. The expression selects all attackers (lines 1-11) and checks that the attackers have at least one dependency (line 12), all their dependencies have a stereotype ≪threat≫ or a subtype (lines 14-20), and attackers are also not assets (lines 21-25).

```
let                                                                                                    1
  stereotype : String = 'Attacker'                                                                     2
in                                                                                                     3
let                                                                                                    4
  attackers : Set (Class) =                                                                            5
  Class.allInstances()->select(                                                                        6
  let                                                                                                  7
      first : Set(Stereotype) = getAppliedStereotypes()->asSet()                                       8
  in                                                                                                   9
  first->union(first->closure(general.oclAsType(Stereotype))). name->includes(stereotype)              10
      )
in                                                                                                     11
attackers->forAll( a | a->asSequence()->first().clientDependency->size() >= 1 )                        12
  and                                                                                                  13
attackers.clientDependency->forAll(                                                                    14
        getAppliedStereotypes().name->includes('controlThreat')                                        15
        or                                                                                             16
        getAppliedStereotypes().name->includes('observeThreat')                                        17
        or                                                                                             18
        getAppliedStereotypes().name->includes('threat')                                               19
        )                                                                                              20
  and                                                                                                  21
  let s : Set(Class) =                                                                                 22
    Class.allInstances()->select(getAppliedStereotypes().name->includes('Attacker') and                23
          getAppliedStereotypes().name->includes('Asset'))
  in                                                                                                   24
s->isEmpty()                                                                                           25
```

Listing 1: *AT01CON* checks that all attackers present at least one threat and are not assets

We use *AE01CON* (see Tab. 1) to check if an *asset* has no relation to the *machine* (in the case of the Common Criteria this is the *ToE*). The reason is that the common criteria certifies products and assets have a relation to that machine. Otherwise, countermeasures in the machine could not protect the assets. If this is the case, this relation has to be added, or the class is not an asset and the stereotype should be removed. We use *ST01CON* (see Tab. 1) to check secondary assets in a similar manner. Moreover, we use OCL to check for missing assets by *AE01REA* (see Tab. 4). The OCL expression *AE01REA* (see Tab. 4) returns all classes that are not an asset and do not contain an asset. Security engineers can reason if this is correct for all the listed assets. This list helps to identify missing assets. For secondary assets, we proceed in a similar way, using *ST01REA* and further expressions in Tab. 1. We introduce our identified OCL expressions that support security reasoning in Tabs. 3, 4, and 5, e.g., by checking for completeness of the threat analysis in domain knowledge diagrams.

3. Consider domain knowledge As a next step, for all assets, either an *assumption* or

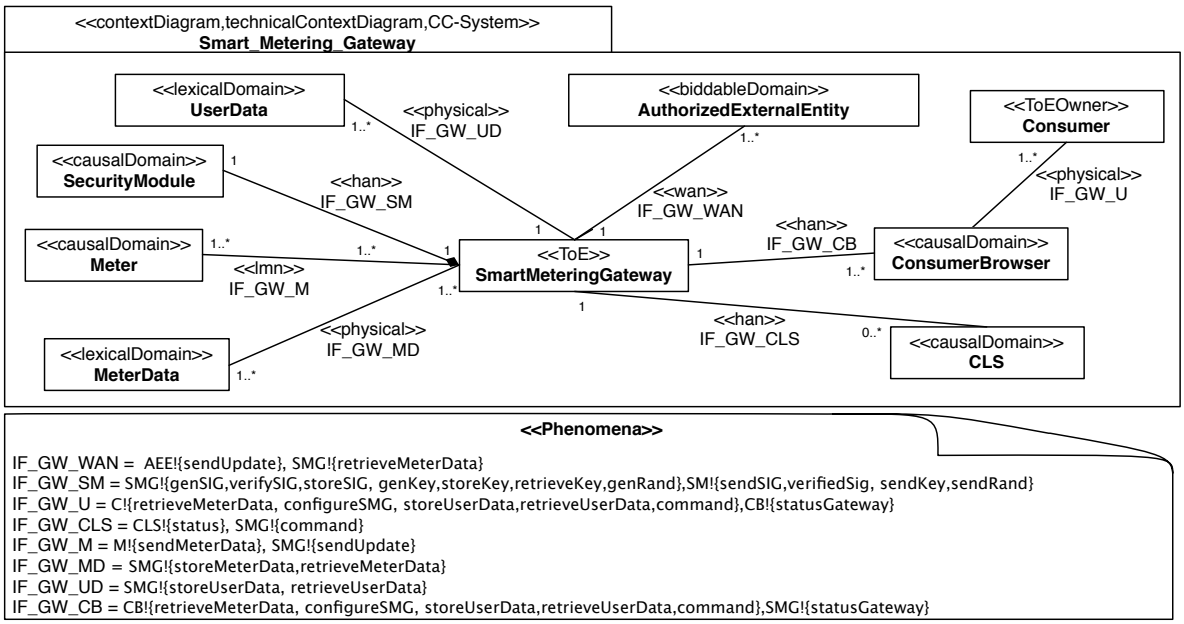

Figure 4: The Context Diagram of the Smart Metering Gateway

fact about its protection has to be described. Facts and assumptions help to estimate if an asset already has sufficient protection and no further security requirement is necessary. They also help to formulate focused requirements that do only address security issues that are not already addressed. In addition, relevant facts or assumptions about assets, which can be exploited by an attacker, have to be documented. Facts and assumptions are documented in domain knowledge diagrams with classes and the stereotypes ≪Fact≫ or ≪Assumption≫ using the UML profile in Fig. 2. The relation between facts and/or assumptions and assets can be documented with dependencies and the stereotype ≪refersTo≫. ≪refersTo≫ states that a statement refers to some domains. It extends the UML meta-class *Dependency*. We use the OCL expressions *FA01CON* and *AS01CON* (see Tab. 1) to check that each fact and assumption ≪refersTo≫ at least one domain. In addition, we use *FA01REA* and *AS01REA* (see Tab. 4) to list all domains having no facts or assumptions (considering all domain knowledge diagrams). For these domains, one should make sure that the most obvious facts have been considered and that facts and assumptions have been distinguished correctly.

4. Describe attackers All attackers have to be described using the attributes shown in Fig. 1. We iterate through all *asset*s and check if they have *assumption*s or *fact*s that prevent them from being threatened by a specific kind of attacker. For example, a piece of software that has no connection with the *ToE* provides no attack vector for a network attacker. Otherwise, an *attacker* has to be introduced that threatens the *asset*. Moreover, the introduced attackers also have assumptions and facts. These have to be modeled explicitly, as well, to support a correct threat assessment.

We use OCL expressions to query our model for getting an overview of all existing threat analysis elements, e.g., assets. These expressions end with the letters "DOC" that stands for documentation. We list all DOC expressions in Tab. 2. For example, expression *AE01DOC* lists all *assets* that can be threatened by an *attacker*. We consider for each asset if an attacker can cause harm to it. Afterwards, we use the expressions *FA01DOC* and *AS01DOC* to check for

each assumption and fact if these can be used to cause harm to an asset. If this is the case, another attacker has to be introduced.

5. Identify threats Threats are a relation between an attacker and an asset. This relation can be modelled with dependencies and the stereotypes ≪threat≫, ≪observeThreat≫, or ≪controlThreat≫. In this step we iterate over all the attackers and introduce threats. *Assumption*s or *fact*s have to be considered or introduced when deciding if the attacker represents an ≪observeThreat≫, or ≪controlThreat≫. We use the *AT01DOC* (see Tab. 2) to list all attackers to start our iteration. Afterwards, we introduce *threats* for each attacker. The OCL expressions *AE01DOC*, *ST01DOC*, *FA01DOC*, *AS01DOC* (see Tab. 2) provide us with lists of domain knowledge artifacts, e.g., facts and assets. After we have introduced threats for the attackers under the consideration of domain knowledge, we check if all attackers represent at least one threat using expression *AT01CON* (see Tab. 1). If an attacker does not represent a threat, the attacker should either be removed, or a threat should be added. We execute the OCL consistency expressions for network, physical, software, and social engineering attacker in a similar fashion (see Tab. 1).

6. Define security objectives Each of the threats leads to the formulation of a security objective. The Common Criteria distinguishes between security objectives (SO), which concern the ToE, and the ones concerning the environment. The latter ones are so-called *security objectives for the environment (SO-OE)*. We model SOs in problem diagrams, because these directed towards the ToE. SO-OEs are modeled in domain knowledge diagrams, because these concern the environment.

7. Validate models and Reason about Security We use OCL to check model consistency of the various diagrams. We defined several validation conditions as OCL expressions, which are listed in Tab. 1. These check, e.g., if facts and assumptions refer to at least one domain. These conditions have to be executed via our support tool in order to validate the models. All existing inconsistencies have to be removed in this step. We have developed debug expressions for all OCL expressions, which state precisely which domain(s) caused the model inconsistency.

In this step we also check for completeness of attackers in the model. For example, the

Table 1: OCL Expressions for ensuring Model-Consistency

OCL-EXPR-ID	Referenced Class	Expression
Domain Knowledge		
FA01CON	Fact	- Refers to at least one domain
AS01CON	Assumption	- Refers to at least one domain
AE01CON	Asset	- Has a relation to the ToE domain (e.g. composition) and is not an attacker
AE02CON	Asset	- Is a domain or part of a domain
ST01CON	Secondary Asset	- Has a relation to an asset and is not an attacker
Threats		
TH01CON	Threat	- Threatens only assets
OT01CON	observeThreat	- Window of opportunity of the attacker is greater than 0
CT01CON	controlThreat	- Window of opportunity of the attacker is greater than 0
Attackers		
AT01CON	Attacker	- Presents at least one threat and is not an asset
NA01CON	Network Attacker	- Threatens only connection domains, connections, or subtypes
PA01CON	Physical Attacker	- Threatens a domain
SA01CON	Software Attacker	- Threatens only causal domains
SE01CON	Social Engineering Attacker	- Threatens only biddable domains

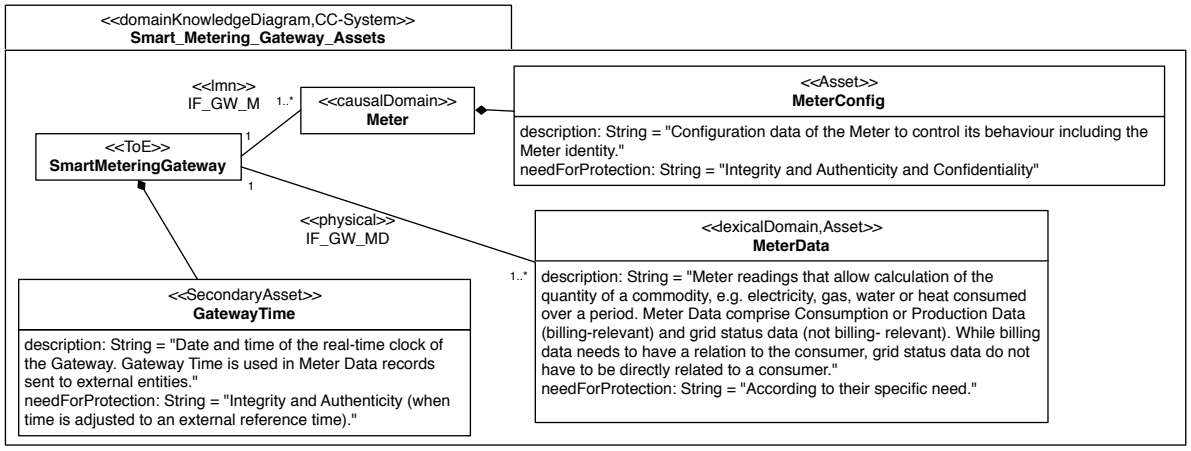

Figure 5: An Example for a Common Criteria compliant asset description

expressions *SA01REA* (see Tab. 3) checks for all causal domains if these are threatened by a software attacker. If this is not the case, we should check for existing assumptions using *AS01CON* (see Tab. 3) for these domains. The resulting information of these expression should serve as a basis for security reasoning for these domains. The question if we need to consider a software attacker for these domains should be answered in particular. The other attacker types are considered in a similar manner using the expressions in Tab. 3. We also use the security reasoning expressions in Tabs. 4 and 5 to reason about the completeness of domain knowledge and threats.

8. Generate documentation Finally, we generate textual documents from the information in the models. Our support tool provides functionalities to query the models and select all relevant information and transform it into text and tables. The output is possible in LaTeX or HTML documents to form the basis for a PP or ST document.

Table 2: OCL Expressions of Document Generation

OCL-EXPR-ID	Referenced Class	Expression
Domain Knowledge		
DO01DOC	Domain	- List all facts and assumptions for each domain
FA01DOC	Fact	- List all facts in the domain knowledge diagrams
AS01DOC	Assumption	- List all assumptions in the domain knowledge diagrams
AE01DOC	Asset	- List all considered assets
ST01DOC	Secondary Asset	- List all considered secondary assets
Threats		
TH01DOC	Threat	- List all considered threats and threatened assets
OT01DOC	observeThreat	- List all considered observe threats and threatened assets
CT01DOC	controlThreat	- List all considered control threats and threatened assets
Attackers		
AT01DOC	Attacker	- List all considered attackers including all attributes
NA01DOC	Network Attacker	- List all considered network attackers
PA01DOC	Physical Attacker	- List all considered physical attackers
SA01DOC	Software Attacker	- List all considered software attackers
SE01DOC	Social Engineering Attacker	- List all considered social engineering attackers

6 Application of our Method

Application scenario We use the protection profile for the smart metering gateway as an example for our method [5]. We apply our method to the creation of a security target and base it on this protection profile. The gateway is a part of the smart grid. The smart grid is a commodity network that intelligently manages the behavior and actions of its participants. The commodity consists of electricity, gas, water, or heat that is distributed via a grid (or network). The benefit of this network is envisioned to be a more economic, sustainable, and secure supply of commodities. Smart metering systems meter the consumption or production of energy and forward the data to external entities. This data can be used for billing and steering the energy production. The "Protection Profile defines the security objectives and corresponding requirements for a Gateway which is the central communication component of such a Smart Metering System" [5, p. 16].

1. Define scope The context diagram shown in Fig. 4 describes the machine to be built in its environment. It is part of the overview description of the security target. The ≪ToE≫ is the SmartMeteringGateway, which serves as a bridge between the Wide Area Network ≪wan≫ and the Local Network ≪physical≫ of the Consumer, the ≪ToE Owner≫. The Meter is connected to the ToE via a Local Metrological Network ≪lmn≫. This is an in-house equipment that can be used for energy management. The Controllable Local System CLS can be, for example, an air conditioning unit or an intelligent refrigerator. The Consumer can also access the ToE [5]

Table 3: OCL Expressions that support Security Reasoning - Attackers

OCL-EXPR-ID	Referenced Class	Expression	Reasoning Support for Security Experts
Attackers			
PA01REA	Physical Attacker	- List all biddable domains that are not threatened by a physical attacker	- Are all humans considered that a physical attacker can threaten?
PA02REA	Physical Attacker	- List all causal domains that are not threatened by a physical attacker	- Are all physical devices considered that a physical attacker can threaten?
SA01REA	Software Attacker	- List all causal domains that are not threatened by a software attacker	- Is every software considered in the threat analysis?
AT01REA	Attacker	- List all attackers that have only observe threats or only controls threats	- Is the attacker's potential modeled correctly?
NA01REA	Network Attacker	- List all connection domains and connections that are not threatened by a network attacker	- Are threats to all relevant domains from that attacker considered?
NA02REA	Network Attacker	- List all connection domains and connections that are not threatened by a network attacker and do not have an assumption	- Are threats to all relevant domains from that attacker considered or do we need to add an assumption?
SA02REA	Software Attacker	- List all causal domains that are not threatened by a software attacker and that do not have an assumption	- If a software attacker is not considered and we do not have an assumption, we should add an assumption or include further software attackers for the resulting domains in the threat analysis.
SE01REA	Social Engineering Attacker	- List all biddable domains that are not threatened by a social engineering attacker.	- Are all possible threats by social engineering attackers considered?
SE02REA	Social Engineering Attacker	- List all biddable domains that are not threatened by a social engineering attacker and that do not have an assumption specified.	- For each biddable domains that is not threatened by a social engineering attacker, we should provide at least an assumption why this is not necessary. If no valid assumption can be found, the threat analysis should be revised to include this attacker.

Table 4: OCL Expressions that support Security Reasoning - Domain Knowledge

OCL-EXPR-ID	Referenced Class	Expression	Reasoning Support for Security Experts
Domain Knowledge			
AE01REA	Asset	- List all classes that are not assets or secondary assets or attackers	- Is an asset still missing?
AE02REA	Asset	- List all assets that have no need-for-protection property	- Is that asset really an asset if it has no need for protection?
AE03REA	Asset	- List all connections or connection domains that do not transmit assets.	- Does a connection between domains really transport no assets?
ST01REA	Secondary Asset	- List all secondary assets	- Are these all really not assets?
DO01REA	Domain	- List all domains that have no facts or assumptions	- Do we really have no domain knowledge at all about a domain?
FA01REA	Fact	- List all domains that have no facts	- Have at least the most obvious facts been considered?
AS01REA	Assumption	- List all domains that have no assumptions	- Have at least the most obvious assumptions been considered?

Table 5: OCL Expressions that support Security Reasoning - Threats

OCL-EXPR-ID	Referenced Class	Expression	Reasoning Support for Security Experts
Threats			
TH01REA	Threat	- List all assets that are not threatened	- Is an asset not threatened at all?
OT01REA	observe Threat	- List all assets that have no observe Threats	- Has an asset only control threats?
CT01REA	control Threat	- List all assets that have no control Threats	- Has an asset only observe threats?

via a ConsumerBrowser. We extended the description for our specification with the following phenomena. The Meter sends meter data to the SmartMeteringGateway. The SmartMeteringGateway stores this data. The Meter can also receive updates from the AuthorizedExternalEntity forwarded via the SmartMeteringGateway. The AuthorizedExternalEntity gets receives meter data in fixed intervals from the SmartMeteringGateway. The SecurityModule provides cryptographic functionalities for the SmartMeteringGateway such as key generation and random number generation. The Consumer can retrieve meter data via the SmartMeteringGateway. The Consumer can also configure the SmartMeteringGateway, send commands to the CLS, receive status messages from the SmartMeteringGateway, and store UserData in it.

2. Asset identification We iterate over the domains in Fig. 4 and identify the MeterData as an ≪asset≫. Figure 5 presents a domain knowledge diagram that contains the description of this asset. The meter data has value for the Consumer, because his/her billing depends upon it and a behavior profile about the Customer can be created from it. Integrity, authenticity, and confidentiality of this data need to be protected. Another asset of the SmartMeteringGateway is the GatewayTime (see Fig. 5). The asset is revealed via investigating assumptions about the SmartMeteringGateway, namely that the meter data is recorded with a correct time stamp. The time is used in MeterData records that are sent to AuthorizedExternalEntity, e.g., for billing. Its integrity and authenticity have to be protected and especially the time adjustment using an externally referenced time is critical.

We use *AE01REA* (see Tab. 4), which returns all classes that are not an asset and do not contain assets. For the smart metering gateway running example, we have so far only

identified the assets MeterConfig, MeterData and GatewayTime (see Fig. 5). The expression *AE01REA* returns: UserData, AuthorizedExternalEntity, CLS, Consumer, ConsumerBrowser, and SecurityModule. For these domains, a good rationale has to be given why they are not assets, or they are have to be marked as assets. For example, the SecurityModule and the connection IF_GW_WAN are indeed assets.

3. Consider domain knowledge The Common Criteria demands that assumptions about domains and connections are made explicit. We choose the assumptions about the AuthorizedExternalEntity, the IF_GW_WAN, and the SmartMeteringGateway as examples (taken from [5]). The assumptions document the assumed behavior of the authorized external entities, reliability and bandwidth of the connection, and the installation location of the SmartMeteringGateway. Assumptions refer to domains in the context diagram. Additionally, facts can be included, e.g., that the SmartMeteringGateway needs electricity to operate. These facts are stated in a domain knowledge diagram, depicted in Fig. 6.

4. Describe attackers We introduce a ≪NetworkAttacker≫, who threatens the WAN connection, depicted in Fig. 6. We have also assumptions regarding this attacker. AssumptionWLANAttacker states that the attacker is located in the WAN and that he/she has the capability to threaten the smart grid, e.g., via sending forged meter data into the grid. This assumption ≪refersTo≫ the WANAttacker. Based upon the AssumptionWLANAttacker we instantiate the attacker with the following attributes: the attacker has the Expertise = Expert (6) and the ElapsedTime = ≤ 1 day (0), the KnowledgeOfToe = Restricted (3), and the WindowOfOpportunity = Unnecessary/Unlimited access (0). We can calculate the value for these attributes for addressing the assurance component AVA_VAN for vulnerability assessment of the CC [10, part 3,p. 16]. The results demand at least an EAL 2 of the CC.

We also know that the Meter depends upon electricity and we introduce the FactElectricity. This can lead to the introduction of a ≪PhysicalAttacker≫. However, the AssumptionPhysicalProtection states that a basic level of physical protection exists. Hence, the introduction of a ≪PhysicalAttacker≫ is not required for this scenario if we install the Meter in locked box. An alternative is to remove the AssumptionPhysicalProtection and consider a sophisticated ≪PhysicalAttacker≫, who can penetrate the physical barriers of the *ToE*.

5. Identify threats The WANAttacker gives rise to the ≪observeThreat≫ T.Disclosure WAN (see Fig. 6), which states that the WAN attacker can disclose meter data or meter configuration data. The WANAttacker causes also the ≪control Threat≫ T.DataModificationWAN, which allows the attacker to modify several different data, e.g., meter data, and meter config data.

We list all threats stated in the protection profile [5] in Fig. 7. The *WANAttacker* causes the following threats related to modification and observation of data:

T.DataModificationWan the unauthorized modification of configuration parameters via using the IF_GW_WAN connection of to the gateway.

T.DisclosureWan disclosing of meter data from the IF_GW_WAN connection.

T.ResidentDataWan the unauthorized reading of data on the gateway that is no longer required and originated from the IF_GW_WAN connection.

T.TimeModification the unauthorized modification of the gateway time.

T.Privacy the IF_GW_WAN connection transports data that is not required by the *AuthorizedExternalEntity*

The *Local Attacker* threatens the *SmartMeteringGateway* with regards to modification and observation of data:

T.DisclosureLocal disclosing of meter data from the IF_GW_CB, IF_GW_M, and IF_GW_CLS connections.

T.TimeModification the unauthorized modification of the gateway time.

T.DataModification the unauthorized modification of configuration parameters using the connections IF_GW_CB, IF_GW_M, and IF_GW_CLS connection.

T.ResidualDataPhysical the unauthorized reading of data on the gateway that is no longer required and originated from the IF_GW_M, and IF_GW_CLS connections.

T.ResidentData the unauthorized modification of configuration parameters via physical access to the gateway.

We use *AT01DOC* (see Tab. 2) to check all considered attackers, which so far only returns the WAN attacker. In order to reason that all relevant attackers have been considered, we execute the following OCL expressions from the security reasoning about attackers listed in Tab. 3. We consider only the domains shown in Fig. 6. *NA01REA* does not list any domain, because the connection IF_GW_WAN is already threatened by a network attacker. *PA01REA* returns the *SmartMeteringGateway*, because a threat caused by physical attacker should be considered. We do not follow this suggestion, because we introduced AssumptionPhysicalProtection in the previous step of our method. *SA01REA* returns that a software attacker should be considered

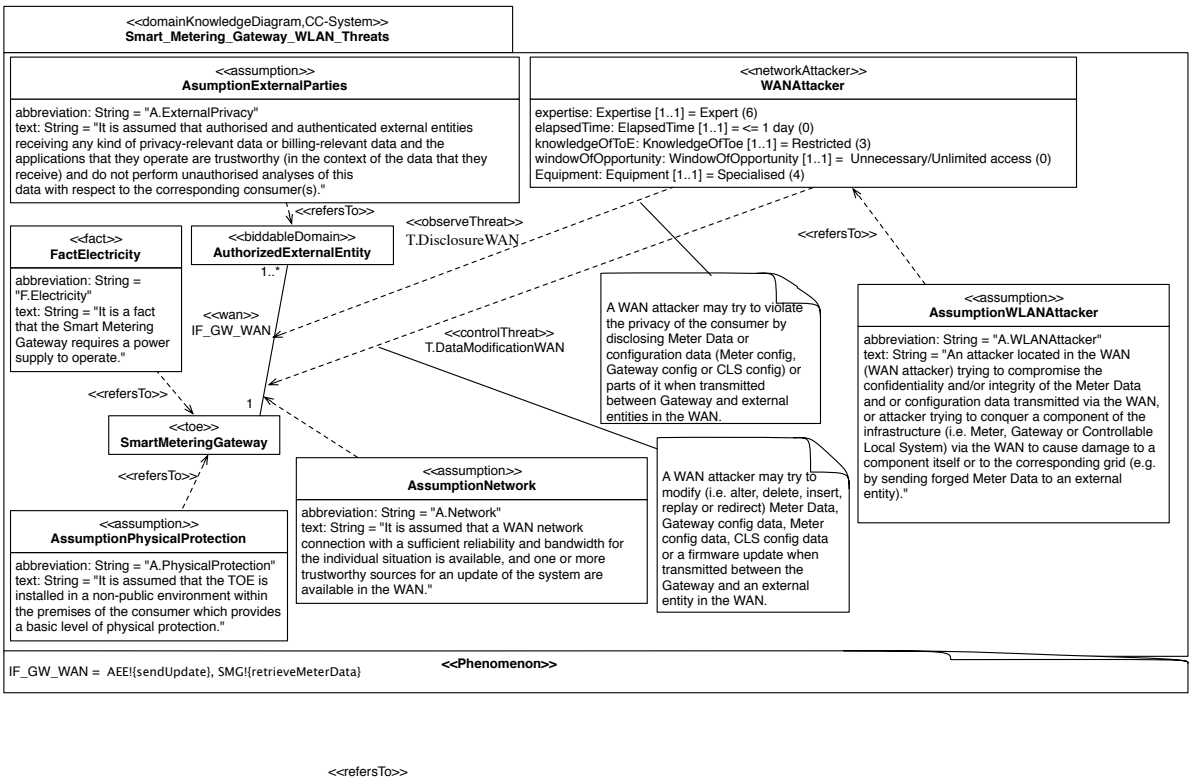

Figure 6: A domain knowledge diagram for Common Criteria compliant threat description

for the SmartMeteringGateway. The software attacker may penetrate the SmartMeterGateway and present a ≪controlThreat≫ towards it. An assumption about the software attacker is that she/he can control the SmartMeterGateway and modify all meter data the gateway has access to. In addition, the OCL expressions *SE01REA* states that a social engineering attacker can threaten AuthorizedExternalEntity. The social engineering attacker presents a ≪controlThreat≫ towards the AuthorizedExternalEntity, and an assumption is introduced that the attacker can control the AuthorizedExternalEntity in such a way that the attacker gains access to the meter data and to the keys and certificates necessary to access the SmartMeterGateway. Hence, the new assumption states that the social engineering attacker can access and configure the SmartMeterGateway. For example, the attacker could use the gateway to control a CLS, e.g., a heater or a refrigerator. This assumption has to be included in Fig. 6, as well as the subsequent threats.

6. Define security objectives We define security objectives in problem diagrams. An example is the security objective *Management* taken from [5, p. 39].

The protection profiles formulates the security objective *Management* as follows:

- The ToE shall only provide authorized Gateway Administrators with functions for the management of the security features.
- The ToE shall ensure that any change in the behavior of the security functions can only be achieved from the WAN side interface. Any management activity from a local interface may only be read.
- Further, the ToE shall implement a secure mechanism to update the firmware of the ToE that ensures that only authorized entities are able to provide updates for the ToE and that

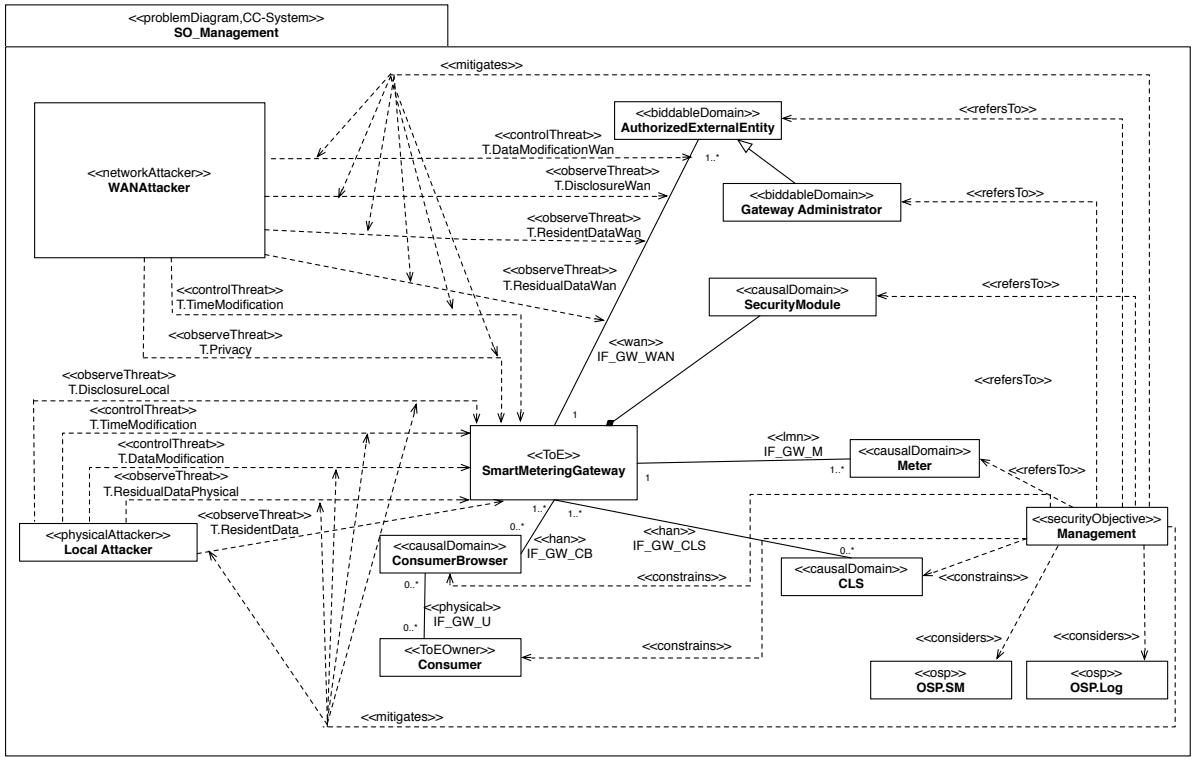

Figure 7: Problem diagram for the security objective "Management"

only authentic and integrity protected updates are applied.

We depict the security objective *Management* in Fig. 7. The security objective *Management* ≪refersTo≫ *Gateway Administrators*, a specific kind of *AuthorizedExternalEntity*. The administrators are the only domain allowed to access the management functionality of the *Smart Metering Gateway*. The functionality for authentication is provided by the *Security Module*, which is described in a separate protection profile [15]. The mandatory usage of the *Security Module* is also defined in the organizational security policy *OSP.SM*, which ≪considers≫ the security objective *Management*. The objective considers the policy *OSP.Log*, as well. This policy states that a set of log files have to be compiled, e.g., for the information flow between the WAN and the smart metering gateway. In addition, all accesses to the configuration data of the *SmartMeteringGateway* are recorded. The security objective *Management* ≪constraints≫ the *ConsumerBrowser*. The *ConsumerBrowser* is not allowed to access the management configuration of the system. The same constraint has to hold for the *Consumer* and the *CLS*. The restriction of access to the management functions of the *SmartMeteringGateway* shall ≪mitigate≫ threats related to modification and observation of data from a *WANAttacker* introduced in the previous step. These are *T.DataModificationWan*, *T.DisclosureWan*, *T.ResidentDataWan*, *T.ResidualDataWan*, *T.TimeModification*, *T.Privacy*.

The restriction of access to the management functions of the *SmartMeteringGateway* shall also ≪mitigate≫ threats related to modification and observation of data from a *Local Attacker* introduced in the previous step, as well. These are *T.DisclosureLocal*, *T.TimeModification*, *T.DataModification*, *T.ResidualDataPhysical*, *T.ResidentData*. lifecycle concerns. The last rows define *Consequences* of the design decisions and *Dependencies* to other parts of the system. to the *AuthorizedExternalEntity*.

7. Validate Models and Reason about Security Table 1 states several conditions that check for consistency problems in the entire model (which contains all context, domain knowledge and problem diagrams). The expression TH01CON checks if all threatened domains and connections are assets. This check fails on our model. One of the issues is that the *WANAttacker* threatens the connection IF_GW_WAN 6, which is not identified as an asset in the protection profile. This is an issue for discussion when improving the protection profile. The connection might not be an asset, because it is not a part of the ToE, but including the connection at least as secondary asset would probably be a good solution.

Tab. 6 shows the use of our OCL expressions for security reasoning. The first column of the table states the expression used. We used the expressions only on a few domains. The second column on the table states the domains considered by the OCL expression. The third column states the results of the query and the last column the resulting security reasoning based on the results of the query.

FA01REA (see Tab. 4) checks if we have modeled facts about domains. This is not the case for the WANAttacker and the AuthorizedExternalEntity. Hence, we have to reason why our assumptions are sufficient and should get feedback on these assumptions from further security experts.

NA01REA (see Tab. 3) queries the model if we have considered a NetworkAttacker for all network connections or connection domains. This is not the case for the network connection IF_GW_CB, which connects the ConsumerBrowser and the SmartMeteringGateway (see Fig. 4). During security reasoning either an assumptions is added to the model instead of a network attacker. We assume that there are no malicious insiders in the ≪han≫, who misuse network traffic. Otherwise a security objective has to be added that states the SmartMeteringGateway has to protect the IF_GW_CB connection by encryption (see Tab. 6).

Table 6: An example for OCL-based Security Reasoning

OCL-EXPR-ID	Class or Relation	Result	Reasoning
FA01REA (see Tab. 4)	SmartMeteringGateway, WANAttacker, AuthorizedExternalParty	WANAttacker, AuthorizedExternal Entity	We do not have facts about the WANAttacker and the AuthorizedExternal Entity. Hence, we have to check if the assumptions documented are valid, e.g., by discussing the results with an independent security expert.
SE02REA (see Tab. 3)	all domains	Consumer, AuthorizedExternal Entity	We have not considered SocialEngineeringAttackers and have no assumptions specified why SocialEngineeringAttackers do not need to be considered. The result of this OCL expression should trigger a threat analysis regarding if SocialEngineeringAttackers are relevant for the Consumer and theAuthorizedExternalEntity.
NA01REA (see Tab. 3)	Consumer, ConsumerBrowser, SmartMeteringGateway, Meter	IF_GW_CB	We have to add a security objective that the communication on the IF_GW_CB connection is encrypted by the SmartMeteringGateway or an assumption that there are no malicious insiders in the ≪han≫, who misuse network traffic.

SE02REA (see Tab. 3) checks if we considered for all biddable domains SocialEngineeringAttackers or have modeled assumptions. For the Consumer and the AuthorizedExternalEntity we have modeled neither. Hence, the result of SE02REA should be discussed in an expert workshop. The experts decide if SocialEngineeringAttackers have to be included into the threat analysis. Alternatively, they have to add assumptions, which explain why the consideration of SocialEngineeringAttackers is not needed.

8. Generate documentation The CC demands a particular description of the *ToE*, which has to follow a specific structure. For example, it starts with an introduction that has to contain a description of the *ToE*, its interfaces, and the operational environment, e.g., the operating system the *ToE* runs on. We show an example how a *ToE* description can be generated using the *UML4PF CC-system extension* and a list of assets. We provide example artifacts for a *ST/PP*: the *ToE-Reference* in the *Introduction* and the *Assets* in the *Security Problem Definition*.

```
1  Property.allInstances()->select(
2  getAppliedStereotypes().name->includes('Asset'))
3  ->collect(c |
4    let st: Stereotype =
5  c.getAppliedStereotypes()->select(name='Asset')->
6          asSequence()->first() in
7      c.name.oclAsType(String).concat(';')
8          .concat(c.getValue(st,'description').oclAsType(String)).oclAsType(String)
9          .concat(';').concat(c.getValue(st, 'needForProtection').oclAsType(String))
10 )
```

Listing 2: AE01DOC - Collecting assets and their attributes.

The *ToE Reference* for a *PP* can be generated from the information collected in the *context diagram* and *technical context diagram*. We use an OCL expression to extract the collected information. Tab. 7 presents the output of the example reference. We collected the information by an OCL expression that collects the information contained in the applied stereotypes ≪CC-system≫ and ≪ToE≫.

We can automatically generate a list of *Assets* using OCL on the information contained in the domain knowledge diagram (see Fig. 5 on page 46). We use the OCL expression AE01DOC (see Tab. 2) and we show AE01DOC in Listing 2. This expression selects all classes, whose name

Table 7: PP Reference - generated by our tool support

Title	Protection Profile for the Gateway of a Smart Metering System (Gateway PP)
Version	01.01.01(final draft)
Date	25.08.11
Authors	Dr. Helge Kreutzmann, Stefan Vollmer (BSI), Nils Tekampe and Arnold Abromeit (TÜV Informationstechnik GmbH)
Registration	Bundesamt für Sicherheit in der Informationstechnik (BSI) Federal Office for Information Security Germany
Certification-ID	BSI-CC-PP-0073
CC-Version	3.1
Keywords	Smart Metering, Protection Profile, Meter, Gateway, PP

contains the String *Asset*. The expression further creates a sequence that contains the *name* of the classes and the values of the attributes *description* and *needForProtection*. These values are separated using a ";". We use our tool support to transform the resulting String to a LaTeXtable (see Sect. 7 for more details on the support tool). The resulting table is shown in Tab. 8.

Table 8: Table for Assets of the PP/ST - generated by our tool support

Asset	**Description**	**Need for Protection**
MeterData	Meter readings that allow calculation of the quantity of a commodity, e.g. electricity, gas, water or heat consumed over a period. Meter Data comprise Consumption or Production Data (billing-relevant) and grid status data (not billing- relevant). While billing data needs to have a relation to the consumer, grid status data do not have to be directly related to a consumer.	According to their specific need.
Consumption Data	Billing-relevant part of Meter Data. Please note that the term Consumption Data implicitly includes Production Data.	Integrity and authenticity (comparable to the classical meter and its security requirements), Confidentiality (due to privacy concerns)
Data / User Data	The terms Data or User Data are used as a hyperonyms for Meter Data and Supplementary Data.	According to their specific need
Supplementary Data	The Gateway may be used for communication purposes by devices in the LMN or HAN. It may be that the functionality of the Gateway, that is used by such a device, is limited to pure (but secure) communication services. Data that is transmitted via the Gateway but that does not belong to one of the aforementioned data types is named Supplementary Data.	Integrity and authenticity (comparable to the classical meter and its security requirements), Confidentiality in the WAN (due to privacy concerns)
Status Data	Grid status data, subset of Meter Data that is not billing-relevant.	Integrity and authenticity (comparable to the classical meter and its security requirements), Confidentiality (due to privacy concerns)
Gateway config	Configuration data of the Gateway to control its behaviour including the Gateway identity and the access control profiles.	Integrity and authenticity, Confidentiality
Firmware Update	Firmware update that is downloaded by the TOE to update the firmware of the TOE.	Integrity and authenticity

Continued on next page

Table 8 – *Continued from previous page*

Asset	Description	Need for Protection
Gateway Time	Date and time of the real-time clock of the Gateway. Gateway Time is used in Meter Data records sent to external entities.	Integrity and Authenticity (when time is adjusted to an external reference time).
CLS config	Configuration data of a CLS to control its behaviour.	Integrity and authenticity, Confidentiality
Firmware	The firmware of the TOE	Integrity, Authenticity
MeterConfig	Configuration data of the Meter to control its behaviour including the Meter identity.	Integrity and authenticity, Confidentiality

The Common Criteria demands a *cross-table* as part of the security objectives section of the ST and PP. The cross table presents the security objectives rational, which analyses the relations between threats, assumptions, organizational security policies and security objectives, as well as security objectives for the environment. The cross-table for our example is depicted in Tab. 9. It shows all security objectives and security objectives for the environment states in the smart metering gateway PP on the horizontal axis and all threats the objectives mitigate on the vertical axis. If an element on the vertical axis is addressed by an objective on the horizontal axis, the box is marked by an "x". For the security rational it is essential that all threats are addressed by at least one security objective or security objective for the environment. Assumptions can only be addressed by security objective for the environment according to the Common Criteria. Organizational security policies have to be considered by at least one security objective or security objective for the environment. If the threats, assumptions, and organizational security policies are addressed reasonably is to be determined by a CC certification body. The cross-table is of utmost importance for a structured argumentation.

We use OCL expression to collect all threats, assumptions, and OSPs. In addition, we collect the relations between these threats, assumptions, and OSPs and the security objectives and security objectives for the environment. We check if the objectives have dependencies with the stereotype ≪mitigates≫ to threats or dependencies with the stereotype ≪considers≫ to assumptions or OSPs. Afterwards, our support tool creates a LaTeX table (see Sect. 7 for more details on the support tool). Table 9 is generated for our example using our support tool.

7 Tool Support

Our method is based on tool support, otherwise the manual creation of all diagrams and manual mapping to textual documents would be very costly in terms of time. We based our tool on the UML4PF tool and named our tool *UML4PF-CC*[2]. Figure 8 shows the architecture of UML4PF-CC. The white boxes in Fig. 8 state components that we implemented specifically for UML4PF-CC and the grey boxes are components that we re-used from the UML4PF tool.

In the following, we list the functionalities and items of UML4PF-CC:

- The *UML Profile for Common Criteria* defines the relevant stereotypes for the Common Criteria, e.g., ≪ToE≫.

[2]This extension is available under the the following homepage:
`http://www.uml4pf.org/cc-extension/index.html`

Table 9: Common Criteria Cross Table - generated with our tool support

	SO.Access	SO.Conceal	SO.Crypt	SO.Firewall	SO.Log	SO.Management	SO.Meter	SO.Protection	SO.SeparateIF	SO.Time	SO_OE.ExternalPrivacy	SO_OE.Network	SO_OE.PhysicalProtection	SO_OE.Profile	SO_OE.SM	SO_OE.TrustedAdmins	SO_OE.Update
T.ResidentData	X			X				X								X	
T.InfrastructureMeter			X					X								X	
T.DisclosureWAN		X		X		X		X	X							X	
T.ResidualDataWAN																X	
T.InfrastructureGateway			X			X	X	X								X	
T.TimeModification						X		X		X			X			X	
T.ResidualDataWan								X									
T.TimeModificationPhysical						X		X								X	
T.InfrastructureCLS			X	X			X	X								X	
T.ResidualDataPhysical						X		X								X	
T.ResidentDataWan								X					X				
T.Privacy				X		X	X	X						X		X	
T.ResidentDataWAN	X																
T.DataModificationWAN				X		X		X								X	
T.DataModificationLocal			X			X	X	X					X			X	
T.DisclosureLocal			X			X	X	X					X			X	
A.AccessProfile														X			
A.ExternalPrivacy											X						
A.Network												X					
A.PhysicalAttacker													X				
A.PhysicalProtection													X				
A.TrustedAdmins																X	
A.Update																	X
A.WLANAttacker											X						
OSP.Log	X				X	X		X								X	
OSP.SM			X			X		X							X	X	

- The *Document Generator* uses the ACCELEO[3] model-to-text transformation tool, which is also an Eclipse plug-in. The *Document Generator* creates HTML and LaTeXdocuments from UML4PF-CC models.
- The *Security Reasoner* contains several OCL expressions that support security reasoning based on UML4PF-CC models, e.g., if all attackers are considered. The reasoner is based on the OCL validator.
- The *OCL Validator* checks if a model is valid and consistent. The *OCL Validator* has been extended to use specific *OCL expressions* for UML4PF-CC models. We contributed these OCL expressions, as well.

[3]The ACCELEO homepage:
`http://www.acceleo.org/pages/home/en`

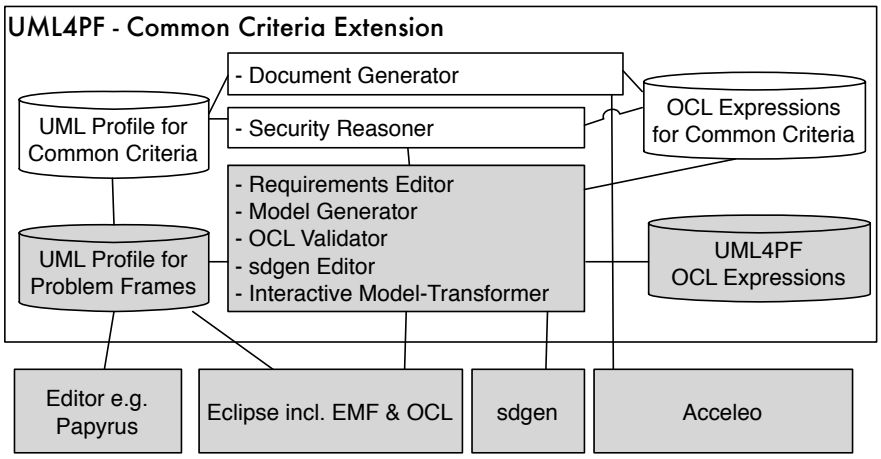

Figure 8: UML4PF common criteria extension support tool

8 Discussion of our Results with Practicioners

Our method was developed based on the experience from several security and especially Common Criteria projects. To illustrate the procedure, we used a case study that creates a Security Target for an existing Protection Profile. The method was discussed with two security consultants, who have already applied parts of the method in industrial projects. In Common Criteria projects, cross-tabulations are created for checking the consistency of documents. Especially the effort for this task is significantly reduced by the presented method and tool. The security consultants also mentioned that this structured procedure

- helps to describe the attackers' abilities in more detail,
- supports to identify all threats to the given assets,
- helps not to forget relevant assumptions or facts, and
- supports to identify and classify assets.

We also asked two evaluators, who check Common Criteria documentations. They responded that they prefer the graphical representation used in our method instead of the plain text and tables in current Common Criteria documents.

The evaluators mentioned the following limitations of our method:

- The amount of text in a class is sometimes distracting.
- The modeling is time consuming.
- The problem frame notation has to be learned beforehand, and
- Our method does not support the entire process of Common Criteria certification.

9 Related Work

We have analzyed related work in the knowledge area of security requirements engineering methods.

Schmidt proposes the problem-frame based method *Security Engineering Process using Patterns (SEPP)* [16], which introduces security concerns into the problem frame notation, e.g., an attacker. The author decomposes the security issues of a system into security problem frames. Each security problem frame concerns a particular security concern of the system in relation to existing functional requirements. The author refines these into concretized security requirements, which also state possible solutions for the security concerns. The author maps the artifacts produced by the SEPP method to the Common Criteria. However, the mapping is proposed, but not shown in an example or integrated into the method itself. Moreover, the SEPP method does not use the CC terminology like ToE and does not support CC-specific document creation.

Mellado et al. [17] created the Security Requirements Engineering Process (SREP). SREP is an iterative and incremental security requirements engineering process. In addition, SREP is asset-based, risk driven, and follows the structure of the Common Criteria. The method uses use cases to model security objectives, and misuse cases to model threats. The authors also developed a template for ranking threats, attacks, and risks. Thy propose a Security Resources Repository (SRR) that can store elicited threats, attacks, and risks. The method differs from our work in the sense that SREP is a method that supports the security reasoning according to the CC. The authors use misuse cases for eliciting threats and their method does not provide clear criteria to decide when all threats are elicited.

Rottke et al. [18] present a problem-driven requirements engineering method for CC compliant systems. This high level method also considers problem frames. The method focuses on creating reliable models for context and problem descriptions. This work differs from ours, because we do not limit our method to context and problem descriptions.

Yin et al. [19] model so-called *early-phase* security requirements with an extended i* model. The authors also describe security policies using a formal model and so-called *late-stage security requirements* in an extended UML model. The extended i* framework adds the modeling element *security flaw*, which can have a relation to goals and soft goals. The goals can be influenced by a threat and eliminated by a security goal, e.g., confidentiality. The policies provide three templates for so-called *stream control*, which specifies rules for network traffic. For example, allowed IP addresses. The extended UML model considers explicitly for each element if it belongs to the ToE, external entities, or communication entities. The method differs from ours, because it focuses on generating CC policies for *stream control*. The method does not aim at providing a holistic support for the generation of *ST/PP* documents.

Abuse Frames are a method for analyzing security issues and the corresponding threats and vulnerabilities by using problem frames [20]. So-called anti-requirements and the corresponding abuse frames are defined. An anti- requirement expresses the intentions of a malicious user, and an abuse frame represents a security threat. In contrast to our method, abuse frames do not consider specific notions of the common criteria and do not support computer-aided security reasoning, e.g., for missing threats.

Mayer et al. present a conceptual model called *Information System Security Risk Management (ISSRM)* [21]. The model defines terms and notions of risk management for IT systems with regard to security and relates this conceptual model to definitions in standards like Common Criteria. ISSRM does not provide a structured method for creating Common Criteria documentation.

Several work focus on ontologies for the Common Criteria.

Bialas [22] introduces an ontology that supports the CC security problem definition (SPD). The SPD contains threats, security policies, and assumptions concerning the ToE. The ontology provides relations between security related elements, e.g., risks and threats. The relations can

be used to create an SPD. For example, the ontology maps specific threats to specific risks. In addition, the ontology can be queried to find countermeasures for specific risks. The author extends the method to a IT security development framework, which is complaint with the CC [23]. The method differs from ours, because the author focuses on creating just the *SPD* and not a holistic support for generating *ST/PP* documents. Nevertheless, Bialas work can complement our own. The stored threats and their relation to the ToE could be implemented as a function to suggest threats in our method.

Chang et al. [24] design an ontology that is intended to decrease the time for CC certification. The ontology supports four different use cases. The first is to query content of the CC standard using a tree. The second use case considers a markup tool that allows the user to mark specific parts of the CC. These marks can contain a choice of predefined comments that can be used to ease the review of CC documents. The third use case considers a CC review tool that can provide a checklist of required documents for evaluating a ToE for a specific EAL. The CC review tool also contains information about required documents. This includes already written and approved documents and documents that have to be revised. The last use case concerns a review report tool. This tool provides an assessment of the review process using the data from the previous use case. This work can complement our own. We could use their ontology after generating documents with our method.

Automated Risk and Utility Management (AURUM) is a method for supporting the NIST SP 800-30 risk management standard [25]. The method is based upon an ontology that supports the elicitation of threats, choosing fitting countermeasures, and calculating risks. In contrast to our work, AURUM focuses exclusively on risk management.

Some work also exists with the aim to improve the Common Criteria or use it as an input for security analysis.

Ardi et al. [26] extend the CC Security Target document with the knowledge of existing vulnerabilities. In particular, the authors add threats from known vulnerabilities to the Security Problem Definition, security objectives from vulnerabilities, and information on how to consider these vulnerabilities in the Security Objectives section. The authors use vulnerability cause graphs and security activity graphs to refine the information from the vulnerabilities. This work can complement our own. We can use the information about existing vulnerabilities in our process as well.

Schneider et al. [27] use organizational learning to check software documentation for relevant parts to elicit security requirements. The basis for the organizational learning software the authors use is the Common Criteria. This work differs from our own, because we aim to create Common Criteria documentation, while the work of Schneider et al. uses the content of the Common Criteria standard to identify relevant parts for security requirements elicitation in software documentation.

We looked in state of the art threat research of security analysis based on the data flow diagrams as proposed by Microsoft [28]. Dhilion [29] models the flow of information in a system and investigates possible interaction points of an attacker with the system. The author proposes to use annotations on the models for security relevant information, e.g., authentication data flows. These annotations are used to check a database for possible threats, but the work does not focus on supporting security standards.

10 Conclusion

We contributed a structured model-based method for threat analysis and security objective elicitation in compliance with Common Criteria. We have extended the UML4PF approach [14], which is based on Jackson's problem frame method [6]. Our threat analysis considers attacker types that threaten specific kinds of Jackson's domains. Thereby, we built on the existing UML4PF tool, and its UML profile for dependability [14]. Our method for problem-based threat analysis and security objective elicitation relies on several OCL expressions, which provide validation, security reasoning and document generation support. Security reasoning is meant in the sense that we can check for completeness of the considered attackers during the threat analysis. Our method includes a structured elicitation, documentation, and validation of assets, assumptions, threats, attackers, and security objectives.

Our method offers the following main benefits:

- A structured process for elicitation of threat analysis elements for a Common Criteria certification
- A tool-supported identification of assets, assumptions, threats, and security objectives
- Support for the reasoning of Common Criteria threats based upon attacker types for Jackson's domain types
- Explicit consideration of domain knowledge in terms of facts and assumptions about attackers, the environment, existing security controls.
- Computer-aided generation of tables and figures for each chapter of a Common Criteria *Protection Profile* or *Security Target*
- Consistency checks of all elements and diagrams of the UML4PF-CC model

We created a UML-based method to support the security analysis and documentation demands of the Common Criteria. Our method is tool supported and has the ability to check models for completeness, validate the models, and generate textual documents from these models. Our method has the potential to introduce model-based analysis for the Common Criteria certification, which is currently based on textual documents and tables. These models can support the discussions about security issues and support a structured threat analysis.

References

[1] K. Beckers, I. Côté, D. Hatebur, S. Faßbender, and M. Heisel, "Common Criteria CompliAnt Software Development (CC-CASD)," in *Proc. of the 28th Symposium on Applied Computing (SAC'13), Coimbra, Portugal*. ACM, March 2013, pp. 937–943.

[2] K. Beckers, D. Hatebur, and M. Heisel, "A problem-based threat analysis in compliance with common criteria," in *Proc. of the International Conference on Availability, Reliability and Security (ARES'13), Regensburg, Germany*. IEEE, September 2013, pp. 111–120.

[3] UML Revision Task Force, "OMG Object Constraint Language: Reference," February 2010. [Online]. Available: http://www.omg.org/docs/formal/10-02-02.pdf

[4] B. Fabian, S. Gürses, M. Heisel, T. Santen, and H. Schmidt, "A comparison of security requirements engineering methods," *Requirements Engineering – Special Issue on Security Requirements Engineering*, vol. 15, no. 1, pp. 7–40, March 2010.

[5] BSI, "Protection Profile for the Gateway of a Smart Metering System (Gateway PP)," Bundesamt für Sicherheit in der Informationstechnik (BSI) - Federal Office for Information Security Germany, Version 01.01.01(final draft), 2011, https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/SmartMeter/PP-SmartMeter.pdf?__blob=publicationFile.

[6] M. Jackson, *Problem Frames. Analyzing and structuring software development problems*, 1st ed. Addison-Wesley, 2001.

[7] UML Revision Task Force, *OMG Unified Modeling Language: Superstructure*, Object Management Group (OMG), May 2010.

[8] D. Hatebur and M. Heisel, "A UML profile for requirements analysis of dependable software," in *Proc. of the 29th International Conference on Computer Safety, Reliability and Security (SAFECOMP'10), Magdeburg, Germany, LNCS*, vol. 6351. Springer-Verlag, September 2010, pp. 317–331.

[9] D. Hatebur, M. Heisel, and H. Schmidt, "A formal metamodel for problem frames," in *Proc. of the 11th International Conference on Model Driven Engineering Languages and Systems (MODELS'08), Toulouse, France, LNCS*, vol. 5301. Springer-Verlag, September-October 2008, pp. 68–82.

[10] ISO/IEC, "Common Criteria for Information Technology Security Evaluation," International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC), ISO/IEC 15408, 2009.

[11] F. Massacci, J. Mylopoulos, and N. Zannone, "Security requirements engineering: The SI* modeling language and the secure tropos methodology," in *Advances in Intelligent Information Systems*, ser. Studies in Computational Intelligence, Z. Ras and L.-S. Tsay, Eds. Springer, January 2010, vol. 265, pp. 147–174.

[12] A. van Lamsweerde, *Requirements Engineering: From System Goals to UML Models to Software Specifications*, 1st ed. John Wiley & Sons, 2009.

[13] M. S. Lund, B. Solhaug, and K. Stølen, *Model-Driven Risk Analysis: The CORAS Approach*, 1st ed. Springer, 2010.

[14] D. Hatebur, *Pattern and Component-based Development of Dependable Systems*, 1st ed. Deutscher Wissenschafts-Verlag (DWV) Baden-Baden, 2012.

[15] BSI, "Protection Profile for the Security Module of a Smart Meter Gateway (Security Module PP)," Bundesamt für Sicherheit in der Informationstechnik (BSI) - Federal Office for Information Security Germany, Version 1.0), 2013, https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/SmartMeter/PP_Security_%20Module.pdf?__blob=publicationFile.

[16] H. Schmidt, "A pattern- and component-based method to develop secure software," Ph.D. dissertation, University Duisburg-Essen, April 2010.

[17] D. Mellado, E. Fernandez-Medina, and M. Piattini, "A comparison of the common criteria with proposals of information systems security requirements," in *Proc. of the 1st Conference on Availability, Reliability and Security (ARES'06), Vienna, Austria*. IEEE, April 2006, pp. 654–661.

[18] T. Rottke, D. Hatebur, M. Heisel, and M. Heiner, "A problem-oriented approach to common criteria certification," in *Proc. of the 21st International Conference on Computer Safety, Reliability and Security (Safecomp'02), Catania, Italy, LNCS*, vol. 2434. Springer-Verlag, September 2002, pp. 334–346.

[19] L. Yin and F.-L. Qiu, "A novel method of security requirements development integrated common criteria," in *Proc. of the 2010 International Conference on Computer Design and Applications (ICCDA'10), Qinhuangdao Branch Qinhuangdao, China*. IEEE, June 2010, pp. 531–535.

[20] L. Lin, B. Nuseibeh, D. C. Ince, and M. Jackson, "Using abuse frames to bound the scope of security problems," in *Proc. of the 12th International Conference on Requirements Engineering (RE'04), Kyoto, Japan*. IEEE, September 2004, pp. 354–355.

[21] N. Mayer, P. Heymans, and R. Matulevicius, "Design of a modelling language for information system security risk management," in *Proc. of the 1st International Conference on Research Challenges in Information Science (RCIS'07), Paris, Ouarzazate, Morocco*. IEEE, April 2007, pp. 121–132.

[22] A. Bialas, "Ontology-based security problem definition and solution for the common criteria compliant development process," in *Proc. of the 4th International Conference on Dependability of Computer*

Systems (DepCos-RELCOMEX'09), Brunow, Poland. IEEE, July 2009, pp. 3–10.

[23] A. Białas, "Ontological approach to the IT security development," in *Internet – Technical Development and Applications*, ser. Advances in Intelligent and Soft Computing, E. Tkacz and A. Kapczynski, Eds. Springer Berlin / Heidelberg, November 2009, vol. 64, pp. 261–269.

[24] S.-C. Chang and C.-F. Fan, "Construction of an ontology-based common criteria review tool," in *Proc. of the 2010 International Computer Symposium (ICS'10), Tainan, Taiwan*. IEEE, December 2010, pp. 907–912.

[25] A. Ekelhart, S. Fenz, and T. Neubauer, "Aurum: A framework for information security risk management," in *Proc. of the 42nd Hawaii International Conference on System Sciences (HICSS'09), Hawaii, USA*. IEEE, January 2009, pp. 1–10.

[26] S. Ardi and N. Shahmehri, "Introducing vulnerability awareness to common criteria's security targets," in *Proc. of the 4th International Conference on Software Engineering Advances (ICSEA'09), Porto, Portugal*. IEEE, September 2009, pp. 419–424.

[27] K. Schneider, E. Knauss, S. Houmb, S. Islam, and J. Jürjens, "Enhancing security requirements engineering by organizational learning," *Requirements Engineering*, vol. 17, no. 1, pp. 35–56, March 2012.

[28] M. Howard and S. Lipner, *The Security Development Lifecycle : SDL : A Process for Developing Demonstrably More Secure Software*, 1st ed. Microsoft Press, 2006.

[29] D. Dhillon, "Developer-driven threat modeling: Lessons learned in the trenches," *IEEE Security and Privacy*, vol. 9, no. 4, pp. 41–47, July 2011.

Author Biography

Kristian Beckers graduated from the University Hamburg in 2008, worked afterwords for the Network Research Division NEC in Heidelberg and the Fraunhofer Institute for Software and Systems Engineering in Dortmund. He is a member of Prof. Heisel's software engineering group at the University Duisburg-Essen since 2011.

Maritta Heisel is a full professor for software engineering at the University of Duisburg-Essen, Germany. Her research interests include the development of dependable software, pattern- and component-based software development, software architecture, and software evolution. She is particularly interested in incorporating security considerations into software development processes and in integrating the development of safe and secure software. She co-authored more than 70 refereed publications, including monographs, book chapters, journals and conference papers. She is member of various program committees and served as reviewer for a number of journals and conferences. Moreover, she is a member of the European Workshop on Industrial Computer Systems Reliability, Safety and Security (EWICS).

Denis Hatebur is a PhD student at University Duisburg-Essen in Germany and since 2004 he is the CEO of ITESYS Institut f. technische Systeme GmbH in Dortmund/Germany. He worked in different industrial engineering projects as a consultancy. In these safety and security projects he was responsible for specification and testing parts. His research interest is the dependability engineering considering requirements engineering, architectural design and testing. In this field he has authored numerous reviewed conference and workshop papers. He holds a Diploma degree in information technology from University of Applied Science in Dortmund/Germany and a Master degree in computer engineering from University Duisburg-Essen.