A Problem-based Threat Analysis in compliance with Common Criteria

Kristian Beckers
University of Duisburg-Essen
paluno
Germany
Email: Kristian.Beckers@paluno.uni-due.de

Denis Hatebur
ITESYS Institute for
Technical Systems GmbH
Germany
Email: d.hatebur@itesys.de

Maritta Heisel
University of Duisburg-Essen
paluno
Germany
Email: Maritta.Heisel@paluno.uni-due.de

***Abstract*—In order to gain their customers' trust, software vendors can certify their products according to security standards, e.g., the Common Criteria (ISO 15408). A Common Criteria certification requires a comprehensible documentation of the software product, including a detailed threat analysis. In our work, we focus on improving that threat analysis. Our method is based upon an attacker model, which considers attacker types like software attacker that threaten only specific parts of a system. We use OCL expressions to check if all attackers for a specific domain have been considered. For example, we propose a computer-aided method that checks if all software systems have either considered a software attacker or documented an assumption that excludes software attackers.**

Hence, we propose a 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. We use UML4PF, a UML profile and support tool for Jackson's problem frame method and OCL for supporting security reasoning, validation of models, and also to generate Common Criteria-compliant documentation. 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. We illustrate our approach with the development of a smart metering gateway system.

***Index Terms*—Common Criteria, Problem Frames, Security Standards, Document Generation, Model-driven Engineering, Security Requirements Engineering**

I. Introduction

The ISO 15408 Standard - Common Criteria for Information Technology Security Evaluation (short CC) - [1] demands a detailed documentation of the software system. This system is a 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 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 [2] and for a CC-compliant threat analysis we require a complete model of the *ToE*.

Goal-based methods, e.g., SI* [3] and KAOS [4], investigate the goals and views of all stakeholders of the system. These approaches 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 have a similar approach, e.g., the asset-driven risk management method CORAS [5] 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 do not represent the complete system-to-be. Thus, we do not use any of these methods for our CC-compliant threat analysis.

The Problem Frames [6] method uses an abstraction of the system-to-be and models the environment of the system around it. Thus, 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 characteristic to determine assets, possible attackers, and subsequent threats for these domains. We show 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 use the benefit of having a complete model of the system-to-be and its environment in domains to conduct a threat analysis.

This means that the results of the threat analysis can be equally complete via reasoning about all possible attackers and threats to domains.

Our main contributions are: a) We propose a structured method to identify and document domain knowledge for assets and threats in terms of assumptions and facts using our CC extension of our UML-profile *UML4PF* [7]. b) We use OCL [8] for security reasoning support, validation of models, and also for document generation as part of our method. We also implement these extension into the *UML4PF* support tool [9].

Our method provides the means to generate texts, figures or tables that can be re-used to create CC documentation. However, the method is generic and can be used for threat

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

analysis during requirements engineering in a given software engineering process.

The remainder of the paper is organized as follows. Section II presents background on Common Criteria and Sect. III. Section III presents the UML profile we use for our threat analysis. Section IV shows our method, and Sect. V illustrates the application of our method to a smart grid scenario. Section VII presents related work, and Sect. VIII concludes and gives directions for future research.

II. 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.

III. UML Profile for problem-based and Common Criteria compliant Threat Analysis

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 [10] notation with stereotypes defined in the UML profile UML4PF [11] 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 (see top of Fig. 1). 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 [12], *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.

Our updated Common Criteria extension for the UML4PF profile is shown in Fig. 1. We introduced a previous version of the profile in [13] and showed a structured software development process for creating Common Criteria documentation during software development. We improved the relations between problem frames and common criteria elements like countermeasures, which are now a kind of domain. In this work we developed a technique for threat analysis compliant to Common Criteria that also relies upon Problem Frame reasoning. *Asset*s have descriptions and a need for protection. Assets have to be a problem frame domain or be part of a domain, because we can not model this in the UML profile directly. We use an OCL expression to enforce this condition (see expression AE02CON in Tab. I). 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.

We extended the CC basic security model. *Threat*s can be further divided into *controlThreat*s and *observeThreat*s. *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*. We propose a threat analysis during require-

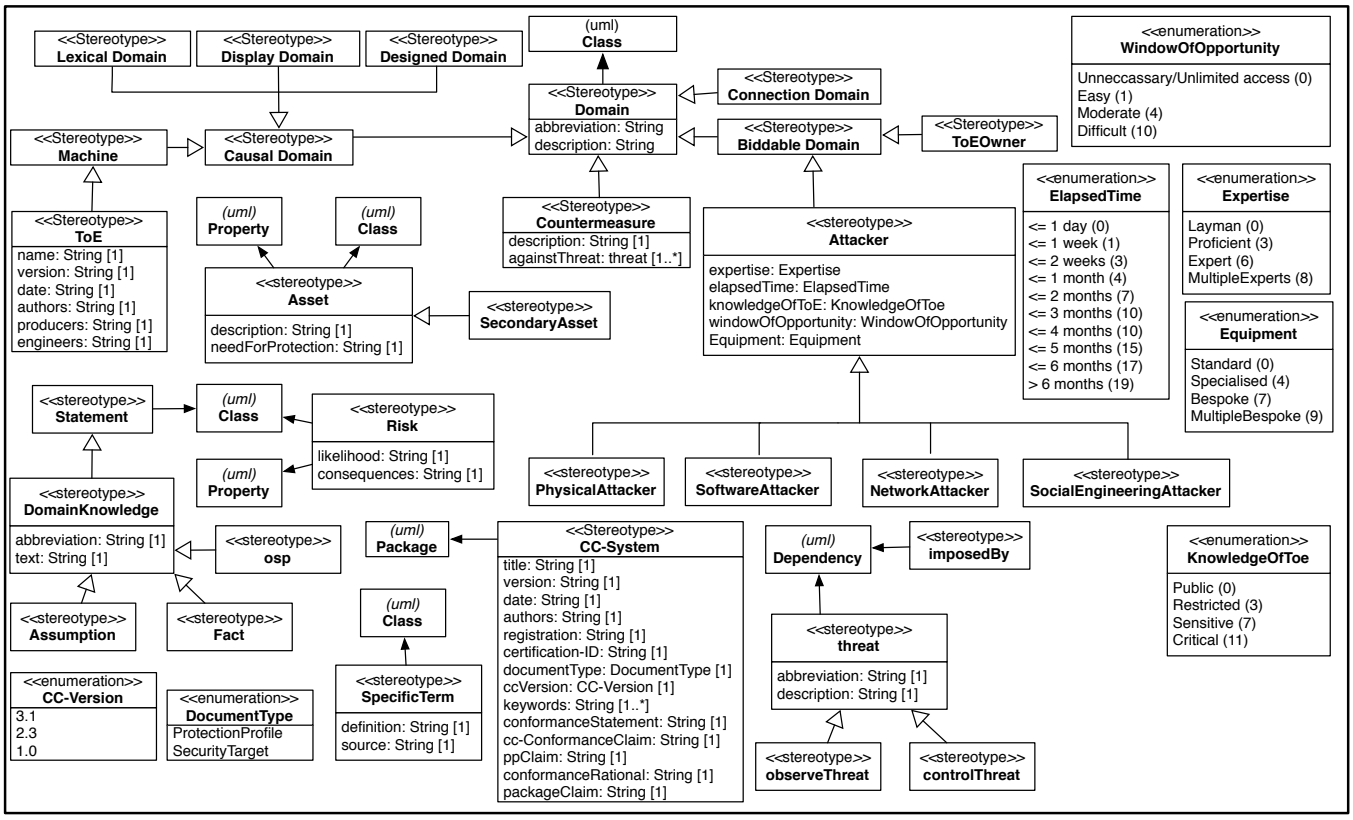

Fig. 1. A Common Criteria extension of the UML4PF Profile

ments engineering, where the exact flow of information is only partially known. Hence, we assume that in the initial threat analysis informational assets can be reached by all domains or interfaces in the model. We model this by allowing threats not only be attached directly to assets, but to all domains or interfaces that allows to reach the asset.

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 *CausalDomains*, e.g., the *ToE* or another software component. *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 and used for the security reasoning.

It is difficult to model behavioral aspects with problem frames. Hence, in our overall method CC-CASD [13] to generate Common Criteria documentation we consider using also behavioral descriptions like UML sequence diagrams. For reasons of space, we do not address this aspect in this paper.

IV. Method for a Systematic Threat Analysis

Figure 2 shows our threat analysis method, which we explain in the following. Grey areas in the figure represent the contribution of this work. 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. I. These expression 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 Fig. 1 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 first selects all special attackers (lines 1 and 2) and checks that all their

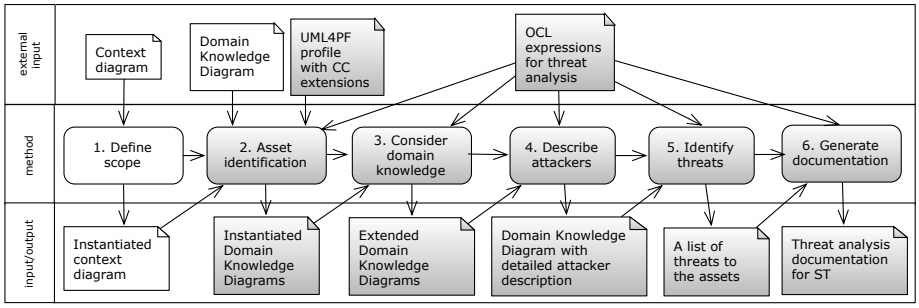

Fig. 2. A Method for Common Criteria-Compliant Threat Analysis; grey areas represent the contribution of this work

TABLE I
OCL EXPRESSIONS FOR ENSURING MODEL-CONSISTENCY OF THE THREAT ANALYSIS

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
Attackers		
AT01CON	Attacker	- Present 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
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

dependencies have a stereotype ≪threat≫ or a subtype (lines 3-5).

```
Class.allInstances()->select(
  getAppliedStereotypes().general.name->includes('Attacker
      '))
.clientDependency->exists(
  getAppliedStereotypes().name->includes('threat') or
  getAppliedStereotypes().general.name->includes('threat'))
```

Listing 1. *AT01CON* reveals all attackers and subclasses that present at least one threat.

1. Define scope To perform the threat 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. 3.

We defined several consistency checks for checking that a context diagram is correct. These are described in detail in [7] 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 a domain, we use UML aggregation or composition relations between the asset and the domain it belongs to. The Common Criteria uses also the notion of *secondary assets*. Attacks to assets cause harm to toe owners, while attacks to secondary assets can cause only harm to assets. In turn harm to secondary assets also causes harm to primary assets and thus to toe owners. An example for a secondary asset is a connection that transports an asset, e.g., a network connection that transports a confidential file.

We use *AE01CON* (see Tab. I) 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 countermeasure 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 also use *ST01CON* (see Tab. I) to check secondary assets in a similar manner. Moreover, we use OCL to check for missing assets.

We introduce OCL expressions that support security reasoning in Tab. II, e.g. by checking for completeness of the threat analysis in domain knowledge diagrams. The OCL expression *AE01REA* (see Tab. II) returns all classes that are not an asset or do not contain an asset.

This list helps to identify missing assets. For secondary assets, we proceed in a similar way, using *ST01REA* and further expressions in Tab. I.

3. Consider domain knowledge As a next step, for all assets, either an *assumption* or *fact* about its protection has to be described. In addition, relevant facts or assumptions

TABLE II
OCL EXPRESSIONS THAT SUPPORT SECURITY REASONING

OCL-EXPR-ID	Referenced Class	Expression	Reasoning Support for Security Experts
Domain Knowledge			
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?
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?
Attackers			
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?
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?
SA02REA	Software Attacker	- List all causal domains that are not threatened by a software attacker and that does 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.
Threats			
TH01REA	Threat	- List all assets that are not threatened	- Is an asset not threatened, meaning secure?
OT01REA	observeThreat	- List all assets that have no observe Threats	- Has an asset only control threats?
CT01REA	controlThreat	- List all assets that have no control Threats	- Has an asset only observe threats?

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 ≪Assumptions≫ using the UML profile in Fig. 1. The relation between facts and/or assumptions and assets can be documented with dependencies and the stereotype ≪refersTo≫. ≪refersTo≫ states that a domain refers to some domains. It extends the UML meta-class *Dependency*.

We use the OCL expressions *FA01CON* and *AS01CON* (see Tab. I) to check that facts and assumptions model at least one domain. In addition, we use *FA01REA* and *AS01REA* (see Tab. II) to list all domains having no facts or assumptions. 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 modelled 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". For simplicity's sake, we do not present a table of all these expression in this paper.

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* to list all attackers to start our iteration. Afterwards we introduce *threats* for each attacker. The OCL expressions *AE01DOC*, *ST01DOC*, *FA01DOC*,

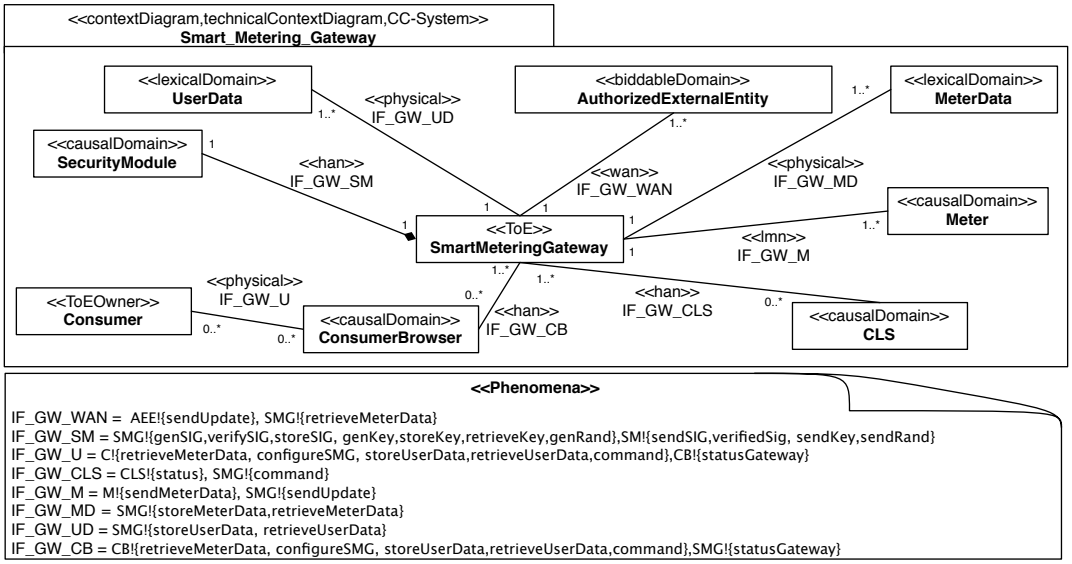

Fig. 3. The Context Diagram of the Smart Metering Gateway

AS01DOC 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. I). 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. I).

In this step we also check for completeness of attackers in the model. For example, the expressions *SA01REA* (see Tab. II) 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. II) 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. II.

6. Generate documentation Finally, we generate textual documents from the information in the domain knowledge diagrams.

We execute all consistency checks (see Tab. I) and check if our model requires improvement. The results of the checks have to be thoroughly investigated. Afterwards we generate Common Criteria documentation as described in [13].

V. Application of our Method

Application scenario We use the protection profile for the smart metering gateway as an example for our approach [14]. 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. This is a commodity network that intelligently manages the behaviour 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" [14, p. 16].

1. Define scope The context diagram shown in Fig. 3 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 for 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 [14] 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 sent meter data in fixed intervals from the SmartMeteringGateway. The SecurityModule provides cryptographic functionalities for the SmartMeteringGateway like 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

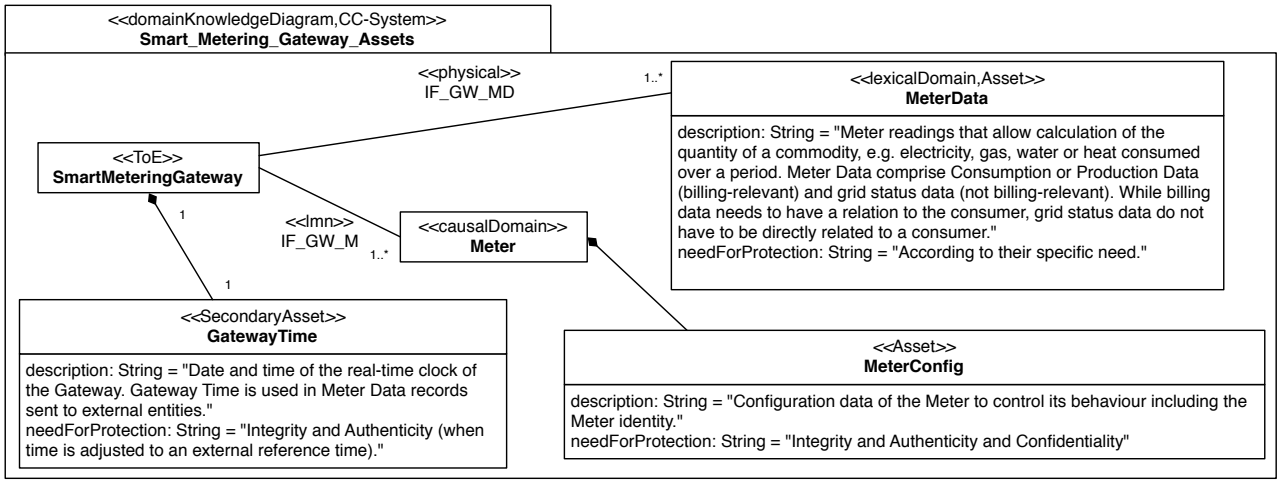

Fig. 4. An Example for a Common Criteria compliant asset description

SmartMeteringGateway and store UserData in it.

2. Asset identification We iterated over the domains in Fig. 3 and identified the MeterData as an ≪asset≫. Figure 4 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 behaviour 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. 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 for, e.g., 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. II), which returns all classes are not an assets 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. 4). The expression *AE01REA* returns: UserData, AuthorizedExternalEntity, CLS, Consumer, ConsumerBrowser, and SecurityModule. For these domains, a good rationale has to be given that these are not assets, or these have to be identified as assets. For example, for the SecurityModule and the connection IF_GW_WAN, assets have to be documented.

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 [14]. The assumptions document the assumed behaviour of the authorised 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. 5.

4. Describe attackers We introduce a ≪NetworkAttacker≫, who threatens the WAN connection, depicted in Fig. 5. 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 of the CC. The results demand at least an EAL 2 of the CC.

We also know that the Meter depends upon electricity and introduce the FactElectricity. This can lead to the introduction of a ≪PhysicalAttacker≫. However, the AssumptionPhysicalProtection states that a basic level of physical protection exist. Hence, the introduction of a ≪PhysicalAttacker≫ is not required for this scenario. If we installed the Meter in a data storage for a bank data center, a sophisticated ≪PhysicalAttacker≫ should be considered, who can penetrate the physical barriers of the *ToE*.

5. Identify threats The WANAttacker gives rise to the ≪observeThreat≫ T.Disclosure WAN (the designation is taken from the PP), 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 use *AT01DOC* 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 *NA01REA*, *PA01REA*, *SA01REA*, *SE01REA* (see Tab. II). These return that a physical attacker should be consid-

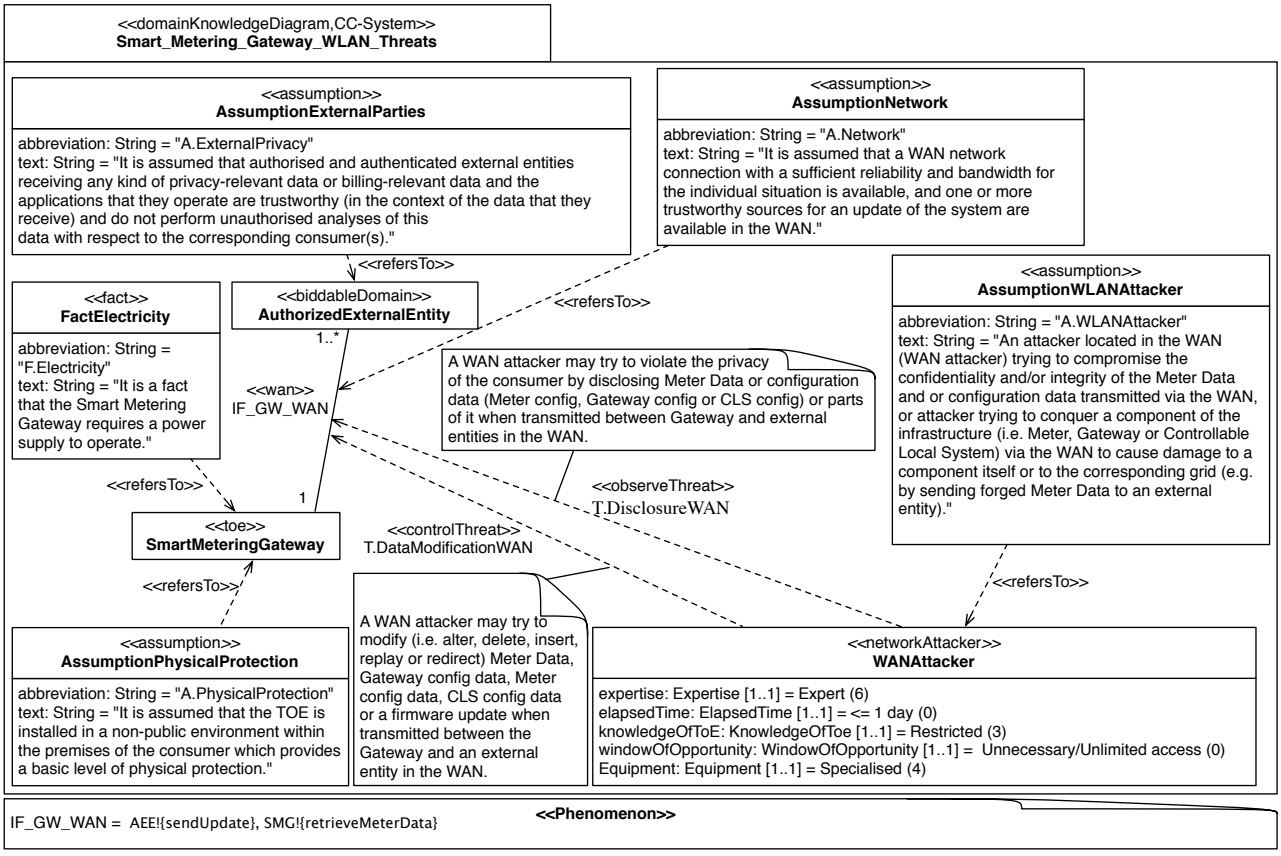

Fig. 5. An Example for a Common Criteria compliant threat description

ered. We do not follow this suggestion, because we introduced AssumptionPhysicalProtection in the previous step of our method. In addition, the OCL expressions return that a software attacker should be considered for the SmartMeteringGateway and a social engineering attacker for the AuthorizedExternalEntity. Both of them have to be modelled in further domain knowledge diagrams and considered in the threat analysis. 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. The social engineering attacker presents a ≪controlThreat≫ towards the AuthorizedExternalEntity and an assumption is 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 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.

Tab. III shows the usage of our OCL expressions for security reasoning. The first column of the table states the expression used. For simplicity's sake, we use the expressions only on a few domains and 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 checks if we have modelled 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 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. 3). During security reasoning two assumptions are added to the model instead of a network attacker. We assume that there are no malicious insiders in the ≪han≫ that misuse network traffic and that the Consumer provides measures to protect the IF_GW_CB connection.

SE02REA checks if we considered for all biddable domains SocialEngineeringAttackers or have assumptions modeled. For the Consumer and the AuthorizedExternalEntity we have modelled neither.

Hence, the result of SE02REA are 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.

TABLE III
AN EXAMPLE FOR OCL-BASED SECURITY REASONING

OCL-EXPR-ID	Class or Relation	Result	Reasoning
FA01REA	SmartMeteringGateway, WANAttacker, AuthorizedExternalParty	WANAttacker, AuthorizedExternalEntity	We do not have facts about the WANAttacker and the AuthorizedExternalEntity. Hence, we have to check if the assumptions documented are valid e.g. by examination of an independent security expert.
NA01REA	Consumer, ConsumerBrowser, SmartMeteringGateway, Meter	IF_GW_CB	We have to add an assumption that the Consumer provides sufficient protection of the IF_GW_CB connection.
SE02REA	all domains	Consumer, AuthorizedExternalEntity	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 threats for the Consumer and theAuthorizedExternalEntity.

6. Generate documentation For reasons of space, we refer to our previous work [13] for the generation of Common Criteria compliant documentation.

VI. DISCUSSION

The procedure presented in this paper was developed based on the experience from several (confidential industrial) security and especially Common Criteria projects. To present the procedure, we use a case study that creates a Security Target for an existing Protection Profile. The method was discussed with 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 the identification of all threats on the given assets,
- helps not to forget relevant assumptions or facts, and
- supports the identification and classification of assets.

We also asked 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 reviewers also mentioned the following limitations.

- 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.

VII. RELATED WORK

Mellado et al. [15], [16] created the Security Requirements Engineering Process (SREP). SREP is an iterative and incremental security requirements engineering process. The approach differs from our work, because the authors use misuse cases for eliciting threats. The approach does not provide clear criteria for when all threats are elicited.

Bialas [17] 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, e.g., risks and threats. The relations can be used to create an SPD. For example, the ontology can provide threats for specific risks. In addition, the ontology can be queried for known countermeasures for these risks. The author extended the approach to the IT security development framework that is complaint to the entire CC [18]. The approach can complement our own. The stored threats and their relation to the ToE could be implemented as threat suggestions in our method.

Ardi and Shahmehri [19] extend the CC Security Target document with a section that considers knowledge of existing vulnerabilities. Their method can complement our own.

Abuse Frames is a method for analysing 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 for, e.g., missing threats.

Schneider et al. [21] 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.

Mayer et al. present a conceptual model called Information System Security Risk Management (ISSRM) [22]. 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.

AURUM is a method for supporting the NIST SP 800-30 risk management standard [23]. 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.

Dhilion models the flow of information in a system and investigate possible interaction points of an attacker with the system [24]. 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.

VIII. Conclusion

We have extended the Jackson's problem frame method with a threat analysis compliant to Common Criteria. Our threat analysis considers attacker types that threaten specific kinds of Jackson's domains. Thereby, we built on the existing UML4PF tool, its UML profile for dependability [7], and an updated versions of its Common Criteria specific extension. Our contribution is a structured method for problem-based threat analysis including several OCL expressions, which provide validation, security reasoning and document generation support. Security reasoning 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, and attackers.

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 and threats
- Support for the reasoning of Common Criteria *threats* based upon attacker types for Jackson's domain types
- Traceability from elements of the *ToE* to threats
- Explicit consideration of domain knowledge in terms of assumptions and facts
- The method can also be used without the common criteria. The UML profile and the OCL constraints can be adapted with little effort to other security standards or methods

In the future, we will work on an extension of the method towards risk management and the selection of countermeasures. We will improve the tool support to allow to hide texts in diagrams and to work towards a holistic modeling support for the Common Criteria. Moreover, we plan to also support further security standards like ISO 27001 [25].

Acknowledgment

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) as well as 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).

References

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

[2] 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, 2010.

[3] F. Massacci, J. Mylopoulos, and N. Zannone, "Security Requirements Engineering: The SI* Modeling Language and the Secure Tropos Methodology," in *AIIS*, ser. SCI, Z. Ras and L.-S. Tsay, Eds. Springer, 2010, vol. 265, pp. 147–174.

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

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

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

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

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

[9] I. Côté, D. Hatebur, M. Heisel, and H. Schmidt, "UML4PF – a tool for problem-oriented requirements analysis," in *Proceedings of RE*. IEEE Computer Society, 2011, pp. 349–350.

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

[11] D. Hatebur and M. Heisel, "A UML profile for requirements analysis of dependable software," in *Proceedings of the International Conference on Computer Safety, Reliability and Security (SAFECOMP) (LNCS 6351)*, E. Schoitsch, Ed. Springer, 2010, pp. 317–331.

[12] D. Hatebur, M. Heisel, and H. Schmidt, "A formal metamodel for problem frames," in *Proceedings of the International Conference on Model Driven Engineering Languages and Systems (MODELS)*, vol. 5301. Springer Berlin / Heidelberg, 2008, pp. 68–82.

[13] K. Beckers, I. Côté, D. Hatebur, S. Faßbender, and M. Heisel, "Common Criteria CompliAnt Software Development (CC-CASD)," in *Proceedings 28th Symposium on Applied Computing*. ACM, 2013, pp. 937–943. [Online]. Available: http://dl.acm.org/

[14] 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.

[15] D. Mellado, E. Fernandez-Medina, and M. Piattini, "A comparison of the common criteria with proposals of information systems security requirements," in *Proceedings of ARES*, 2006, p. 8 pp.

[16] D. Mellado, E. Fernández-Medina, and M. Piattini, "Applying a security requirements engineering process," in *ESORICS 2006*, ser. LNCS 4189, D. Gollmann, J. Meier, and A. Sabelfeld, Eds. Springer Berlin / Heidelberg, 2006, pp. 192–206.

[17] A. Bialas, "Ontology-based security problem definition and solution for the common criteria compliant development process," in *Dependability of Computer Systems*, vol. DepCos-RELCOMEX '09. Fourth International Conference on, 2009, pp. 3–10.

[18] 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, 2009, vol. 64, pp. 261–269.

[19] S. Ardi and N. Shahmehri, "Introducing vulnerability awareness to common criteria's security targets," in *Software Engineering Advances, 2009. ICSEA '09. Fourth International Conference on*, 2009, pp. 419–424.

[20] L. Lin, B. Nuseibeh, D. C. Ince, and M. Jackson, "Using abuse frames to bound the scope of security problems," in *RE*, 2004, pp. 354–355.

[21] K. Schneider, E. Knauss, S. Houmb, S. Islam, and J. Jürjens, "Enhancing security requirements engineering by organizational learning," *Requirements Engineering*, vol. 17, pp. 35–56, 2012.

[22] N. Mayer, P. Heymans, and R. Matulevicius, "Design of a modelling language for information system security risk management," in *RCIS*, 2007, pp. 121–132.

[23] A. Ekelhart, S. Fenz, and T. Neubauer, "Aurum: A framework for information security risk management," in *HICSS*, 2009, pp. 1–10.

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

[25] ISO/IEC, "Information technology - Security techniques - Information security management systems - Requirements," International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC), ISO/IEC 27001, 2005.