Complex Systems Informatics and Modeling Quarterly (CSIMQ)
eISSN: 2255-9922
Published online by RTU Press, https://csimq-journals.rtu.lv
Article 179 , Issue 32, September/October 2022, Pages 28–43
https://doi.org/10.7250/csimq.2022-32.02

Security Requirements Specification and Tracing within Topological Functioning Model

Erika Nazaruka*

Department of Applied Computer Science, Riga Technical University, 10 Zunda Embankment, Riga, LV-1048, Latvia

erika.nazaruka@rtu.lv

Abstract. Specification and traceability of security requirements is still a challenge since modeling and analysis of security aspects of systems require additional efforts at the very beginning of software development. The topological functioning model is a formal mathematical model that can be used as a reference model for functional and non-functional requirements of the system. It can also serve as a reference model for security requirements. The purpose of this study is to determine the approach to how security requirements can be specified and traced using the topological functioning model. This article demonstrates the suggested approach and explains its potential benefits and limitations.

Keywords: Security Requirements, Requirements Traceability, System Modeling, Topological Functioning Model, Reference Model.

1 Introduction

Security requirements refer to providing confidentiality, integrity, and availability of information assets. Over time, the concept of integrity of data has transformed into the concept of trustworthiness of data [1]; and the confidentiality of data has become more focused on the requirement of data privacy. Security requirements derive from domain-related and technology-related legal and regulatory documents. As Liu mentions [1], for handling security requirements, several groups of modeling and analysis approaches exist – goal-based, scenario-based, semi-formal, and formal models as well as ontologies and patterns. Each of the groups has its own purpose and advantages in certain cases.

Security requirements analysis and specification is a very specific issue. It has no large elaboration with the Model-driven Architecture (MDA). The MDA is a set of principles, models, and viewpoints that are dedicated to more formal and automated development of software systems. The main principle applied within the MDA is a separation of concerns. Thereof, each model reflects a viewpoint on the system from one certain concern: computation independent,

* Corresponding author

Reference: E. Nazaruka, "Security Requirements Specification and Tracing within Topological Functioning Model," Complex Systems Informatics and Modeling Quarterly, CSIMQ, no. 32, pp. 28–43, 2022. Available: https://doi.org/10.7250/csimq.2022-32.02

Additional information. Author's ORCID iD: E. Nazaruka – https://orcid.org/0000-0002-1731-989X. PII S225599222200179X. Received: 15 June 2022. Revised: 4 October 2022. Accepted: 9 October 2022. Available online: 28 October 2022.

platform independent, and platform specific. Requirements for the system and a domain model are linked with the computation-independent viewpoint and, correspondingly, model. How to represent security requirements within it is a question. Two opportunities, potentially, are visible: either as a separate model or as references to an external element (i.e., a requirement). This article focuses on the second opportunity.

The goal of this research is to evaluate the suitability of a Topological Functioning Model (TFM) for referencing security requirements as well as perspectives of this model to be used as an aid for the analysis of potential security threats. This article continues the research started earlier [2], [3] and focuses on referencing security requirements. Previous articles ([2] and [3]) focused on demonstrating the capabilities of the TFM as a central reference model for model-based software development. Thanks to the formal mathematical background of the TFM and mappings between the TFM and functional requirements and the TFM and elements of a logical design model, it was possible to extrapolate the same reference principles to the non-functional requirements, such as performance requirements. However, [2] and [3] did not address the main issue of this article about references to security requirements.

To understand the suitability of the TFM for referencing security requirements, the following research methodology was used:

1. First, the systematic literature review on security requirements traceability from specification to logical design models to code elements (together with other functional and non-functional requirements) is done. For having more complete coverage of the existing approaches, the following search criteria were used:
   - Language – English;
   - Publishing year – approximately last 15 years, i.e., from 2007 to 2022 (including);
   - Keywords – from combinations of at least two categories: general and domain, domain and specific;
   - Publication database – IEEE Xplore Digital Library.

The found information sources were filtered additionally by their relevance to the research questions (Section 2). As a result, the classification of the traceability approaches with their strengths and weaknesses was created.

2. Second, the suitability of the TFM for referencing security requirements forward and backward was analyzed theoretically. As a result, the proper internal structure of the model was refined.
3. Third, the application of the theoretical results was illustrated and the validity of the results was explained.

The article is organized as follows. Section 2 provides a short overview of the related work on the specification of security requirements. Section 3 gives, shortly, the background information about TFM and its use as a formal reference model. Section 4 describes referencing security requirements using the TFM. Section 5 illustrates the proposed approach and its properties. Reflections on the applicability of the TFM for referencing security requirements conclude the article.

2 Related Work on Security Requirements Specification and Traceability

The issue of security requirements specification in analysis and design models has existed for a long time and has not lost its importance after the appearance and evolution of the MDA and model-driven development. There are multiple approaches for the analysis and design of security requirements, where security can be an entire single object or a part of the object of the analysis and design [4]. Security requirements may be specified separately or as a part of a software model. Proper specification and established forward/backward traceability mechanisms can make verification of security requirements and change impact analysis more predictable. The open questions analyzed in related works are the followings:

- Q1. What are the benefits and limitations of different approaches for the specification of security requirements including traceability mechanisms?
- Q2. How forward traceability from security requirements to the design model to code elements is implemented, and what are its benefits and limitations?
- Q3. How backward traceability from code elements to the design model to security requirements is implemented, and what are its benefits and limitations?

The publication database, mentioned in the introduction, was searched for scientific publications on security requirements referencing (specification and traceability). The keywords used were the following: *((security requirement* AND model* AND referenc*) OR (security requirement* AND traceability) OR (security requirement* AND tracing) OR (security requirement* AND specification) AND (security requirement* AND verification))*, where "*" represents any possible ending of the word. In the beginning, 650 papers were returned by this request, 58 of them were found relevant by the title, 47 of them where found relevant by the abstract, and first 10 of them were found to be relevant by the content quality and presented details were analyzed. To extend the number of information sources about the use of reference models, the request *((security requirement* AND model* AND referenc*)* was searched by Google and found information sources were filtered by relevance, content, and presentation quality. As a result, 4 more papers were reviewed. The considered sources cover those specification, referencing and traceability approaches that are most commonly cited.

2.1 Benefits and Limitations of Specification Approaches

Approaches for security requirements specification may be grouped into formal, semi-formal and informal ones.

Formal approaches. The formal approaches are the most non-ambiguous, since they apply mathematical logic or programming language principles. They allow using descriptive text in a natural language only as a reference to the original formulation of the security requirement. Thus, the approach presented in [5] in 2022, represents requirements as code, namely, as object-oriented (OO) class templates. The presentation as OO classes benefits from OO principles: generalization and inheritance (eliminated redundancy), improved maintainability, combinations with formal notations (logic), OO analysis, and integration with DevOps tools and processes. However, many open questions on specification of multiple quality attributes, types of security requirements (positive/negative, functional/non-functional), and unclear applicability for industrial case studies still exist. Besides, such specification is far from a human-readable format. Another approach, the UML-like approach with logic, presented in [6] in 2018, uses first-order logic and meta-modeling for the definition of the main principles of specification of security requirements within the model. The security requirements are specified as properties that supplement the modeled system. Corresponding security policies are assigned to the properties. The main drawback, mentioned by the authors of this approach, is a lack of explicit traceability from requirements to model elements to code.

Despite the formalism used, complexity of such specifications requires additional knowledge of developers and manual or partially automated activities for requirements in natural language formalization and their dependency discovery.

Semi-formal approaches. Semi-formal approaches use a combination of formal means and natural language, where the latter plays a supplementing role for different diagrams or models. Therefore, the required information can be kept as within as separately from models/diagrams. Semi-formal approaches form the largest group of approaches. Below, they are analyzed in chronological order.

From 2006 to 2009. One representative of in-model specification is the security requirements engineering method where use cases and misuse cases are combined together in order to make proper analysis and modeling of threats, attacks, and risks presented by Mellado *et al*. [7] in 2006. The benefits of this method relate to using understandable notation and means of the

requirements engineering. Such specification is well-observable but lacks a representation of the sources of security requirements.

From 2010 to 2019. The method proposed by Yin and Qiu [8] in 2010, suggests that security requirements will be specified at three levels (or steps of the method): in the *i**-model, as formally specified policy, and as a use case scenario in UML. This means that here security requirements will be incorporated into the model of the system.

Tøndel *et al*. (2010) explain the use of UML activity diagrams and use case models for specification and analysis of misuse cases [9]. The general idea is to combine misuse cases with attack trees and security activity models thus improving the analysis of potential attacks and threats. In reality, the security model is integrated with certain parts of the system model. Thus, this approach also can be considered as an in-model specification.

Nhlabatsi *et al*. [10] in 2015, presented an approach that specifies security requirements in natural language. Then, these requirements are related to security controls by explicit traceability links called causal traces. The approach is based on the correct definition of assumptions. The weaker assumptions are discarded. This example can be considered as a non in-model specification of security requirements.

Zhioua *et al*. [11] in 2017 published their vision on security requirements. In their approach security requirements specified in natural language are manually transformed into a formal representation (similar to Java programming language) by a security expert. The security expert extracts key elements and builds formulas and patterns on the chosen formalism. Security guidelines and requirements can be modeled in form of a sequence of atomic propositions or statements that represent the behavior of the system. The program is modeled by using the Program Dependence Graph which represents both control and data dependencies. The mapping between the abstract propositions and the program model is managed in the security knowledge base. The main weaknesses of the approach are manual activities: manual extraction of key elements from security requirements and guidelines; and manual establishing of mappings among elements. Since the proposed approach is manual, it lacks explicit traceability. This example can be considered as a non in-model specification of security requirements.

One more interesting in-model approach is presented by Ramadan *et al*. in 2017, where the authors describe the use of Business Process Model and Notation (BMPN) for the specification of design-level security verification [12]. High-level requirements are specified in SecBPMN2 language and translated to secure architectural models in UMLsec. SecBPMN2 uses 11 security annotations to BPMN elements. In turn, UMLsec is the UML supplemented with security specific “stereotypes” and “tags”. Automated acquisition of code is possible in the case of the development of proper transformation modules. The authors highlighted that their framework can be used for automatically establishing traceability between high-level security requirements and technical security policies and the flow of threatening activities. The use of BPMN allows negotiating with the domain experts; and automatically obtained UML sequence diagrams allow proper implementation by the developers. However, human errors can be entered at the very beginning of modeling.

Another approach, presented by Emeka and Liu in 2018, considers requirements specification using controlled formal language SOFL (Structured Object-oriented Formal Language) with mandatory pre and post conditions for dependency analysis and CDFD (Conditional Data Flow Diagrams) automated construction. However, the manual transformation of textual requirements into SOFL and then from CDFD to attack trees requires experts participation. This example can be considered as a non in-model specification of security requirements.

From 2020 to 2022. Ponsard *et al*. [13] in 2020, presented their vision of using a KAOS model. Security and safety requirements are expressed using principles of Goal Oriented Requirements Engineering. Mappings between a KAOS model and constraint programming elements are set manually. The formal semantics is based on temporal logic and a formal pattern library. The approach allows automated analyzing of multi-parameters of security in high-level design models. The main limitations are the granularity and rounding of calculation that may

cause the incorrectness of mappings as well as one-level traceability. This example can be considered as an in-model specification of security requirements.

Quamara *et al*. proposed in 2021 a modeling framework for security requirements [14] that has three layers: a mission layer that is used for capturing *what* is needed to be achieved by the system; a functional layer that is used for capturing *how* to achieve *what* is needed to be achieved by the system; and an architectural layer that is used for capturing *which* elements can finally realize the *"what"* and *"how"*. In such a way formalization of both model and security properties is achieved. Semi-formal constructs of UML are used for graphical representation of the architecture, functions, and mission. Formal constructs are used for specification, reasoning, and verification of properties as First-Order Logic expressions but are not limited to them. This example can be considered as an in-model specification of security requirements.

Tsoukalas *et al*. [15] in 2021, presented their vision of the Security-by-design approach. In this approach, textual security requirements are processed using Natural Language Processing techniques defining such classifiers as Project, Priority, Security Characteristic, Action, Actor, Object and Property. Class Requirement and sub-class structures for classifiers are used. Each requirement is represented as a JSON object. Syntactical and semantical similarities are analyzed for classes with the same (or having similar meaning) classifiers using the ontology WordNet. One of the benefits is the ability to create the Security Requirements Knowledge base and use it for automated searching for similar or additional requirements and suggestions of alternatives. The main drawback is that this approach requires the participation of security experts for results validation. Besides, it cannot handle interconnected, overlapping, and redundant requirements. This example can be considered as a non in-model specification of security requirements.

Olthuis *et al*. [16] in 2021, presented their approach that uses generated (generic) traces. In this approach, requirements in natural language are manually transformed into formal language LTL stored in JSON files. Then, traces are generated in the common trace format (CTF). Such representation allows executing, design, and verifying requirements prior to their implementation. However, the representation of requirements and traces are not user-friendly. This example can be considered as a non in-model specification of security requirements.

Semi-automatic semantic and probability analysis is presented by Wang *et al*. in 2022 [17]. Here, high-level security requirements can be traced horizontally and vertically in an automated way by using manually determined "indicators" of input, time, task refinement, triggering conditions, and realization similarities with rather high recall value. The indicators are reusable. The manual identification of indicators in high-level security requirements and the focus only on textual requirements are the main mentioned weaknesses. This example can be considered as a non in-model specification of security requirements.

As rightly noted in the article [12], BPMN and UML based approaches address security in the development phases without a proper alignment to sources of security requirements and to the requirements themselves. Both approaches – the in-model specification of security requirements and referring to the externally kept security requirements and their sources – are useful. The advantages of the in-model specification are the direct link with the functional parts of the system and the possibility to "execute" the specification before its implementation. The out-of-model specification also has its advantage, i.e., the specification of security requirements themselves may be integrated with threats, vulnerabilities, attack scenarios, risks, sources, and other factors.

Looking at the tendencies before 2020, one can see that UML and BPMN modeling principles were used in most. Starting from 2020, one can conclude that today's trend is on supplementing (or even replacing) modeling activities with Artificial Intelligence techniques, such as Natural Language Processing and Machine Learning Models. They allow elimination of manual activities of security experts on dependency analysis among security requirements and on verification of security requirements based on their descriptions, accepted security procedures, guidelines, and standards.

Informal approaches. Informal approaches are well-elaborated now and wide-spread in the IT industry, therefore are less presented among the scientific research results. They include such techniques as manual layering, key indicators, and manual informal specification and analysis of security requirements and trace links by using predefined security procedures and checklists. Their main weakness is that they require a long time for analysis of security requirements, their interdependencies and their impact on other related elements. As an example of such approaches, the classification of most reusable security requirements sources presented by Schmitt and Liggesmeyer [18] can be considered. It includes three elements: security information and knowledge (diagnostic and prescriptive), software requirements engineering (SRE) methods, and compliance obligations. The first two sources provide knowledge about threats, weaknesses, and vulnerabilities. The last source imposes raw requirements. All threats, weaknesses, vulnerabilities, and raw requirements must be analyzed together when specifying security requirements. The authors propose to define security requirements scope areas and then re-use those topic specific fragments from the sources during the analysis and specification of security requirements. In essence, the authors suggested creating a repository where all the sources of security requirements are organized in scope areas and each scope has sources from all the three categories.

2.2 Forward and Backwards Traceability

Not all of the considered research papers included detailed enough information on forward and backward traceability. There are different types of traceability mechanisms (Table1): direct tracing using OO principles for security requirements formalized specifications [5]; mapping rules in trace models [10], [12]; or theoretically proven mappings from design to programing language elements [15]. Most of the traceability mechanisms implement forward trace links from the security requirements origins (sources) to the formal representation or design elements. Just those using OO principles are forward traceable to code constructs [5], others are theoretically possible but exact implementation was not presented [10], [5] due to the focus of research on requirements verification and dependency analysis.

Table 1. Forward traceability mechanisms and their limitations

Approach	Year	Mechanism	Limitations
Requirement as Code [5]	2022	Direct tracing via implementation	A difficult-to-read format for non-technical stakeholders.
Semantic and probability analysis [17]	2022	Security requirements can be traced horizontally and vertically; five types of dependencies;	Manual identification of indicators in high-level security requirements. Implementation of forward traceability to code elements is absent.
Security-by-Design [15]	2021	Theoretically it is possible to determine implementing classes by strict constructs actor-action-object.	Implementation of forward traceability to code elements is absent.
Generic traces [16]	2021	Matching traces are explicitly represented in a JSON file and match trace events to abstract propositions in the design specification.	Implementation of forward traceability to code elements is absent.
SecBMPN2 and UMLsec [12]	2017	Traceability (mapping rules in trace models) between high-level security requirements and verifiable technical security policies as well as, potentially, to code elements.	Forward traceability to code elements is possible in the case of code generation.
Causal traceability [10]	2015	Causal traces from the source artifact to the specification.	Implementation of forward traceability to code elements is absent.

Looking at the same research for backward traceability, we see that in several cases the mechanism used differ (Table 2). Direct tracing allows using the same principles in forward and backward directions and has the same issues [5]. In the case of using trace models [11], backward traceability is possible when using trace logs of transformations and transformation modules are available. In the case of using some set of indicators for requirements verification or dependency analysis, backward traceability to the source of the requirement is established, but between different levels of implementation is not presented.

Table 2. Backward traceability, its benefits and limitations

Approach	Year	Mechanism	Limitations
Requirement as Code [5]	2022	Direct tracing via implementation.	A difficult-to-read format of security requirements for non-technical stakeholders
Semantic and probability analysis [17]	2022	Five types of dependencies.	Only dependencies at one level of security requirements
Security-by-Design [15]	2021	Reference to the initial requirement in natural language is explicit.	Backward traceability from code constructs is absent.
Generic traces [16]	2021	Matching traces are explicitly represented in JSON file and matches trace events to abstract propositions in design specification.	Backward traceability from code constructs is absent.
SecBMPN2 and UMLsec [12]	2017	Traceability (mapping rules in trace models) between high-level security requirements and verifiable technical security policies as well as, potentially, from code elements.	Backward traceability from code elements is possible in the case of using automated transformation modules and tracing logs.
Causal traceability [10]	2015	Causal traces from the specification to the source artifact.	Backward traceability from code constructs is absent.

In general, there are many approaches that researchers develop and suggest for security requirements specification and analysis to keep the relation with the original requirements or sources of those requirements, formalize specification of requirements in natural language in order to find inconsistencies among them and/or to implement them in design and code elements. The main focus now is on the source-requirement-design chain. Specification of security requirements concerns also behavioral (functional) characteristics of the system and links security-related limitations to them. Specification of those links can be implemented as an in-model or out-of-model solution. However, establishing traceability links from security requirements sources to code elements and backward is less developed. The largest part of the research focuses on the "source- formal specification" path and rarely to the "design element". Besides, tracing from design elements to code elements is not described.

In this article, the application of TFM as a reference model and traceability principles of topological functioning modeling are described. The presented approach is one of the in-model specification approaches and inherits the same benefits and limitations. However, thanks to the formal and holistic nature of the TFM, it includes also "trace models" between informal specifications and formal elements of analysis, design, and implementation. It opens an opportunity to leverage the advantages of element direct tracing.

3 TFM as a Reference Model for Requirements

This section describes the main elements of the TFM and the main concepts of the topological functioning modeling starting from the essential TFM constructs and ending with known capabilities of this model for transformation into other software development artifacts.

3.1 Topological Functioning Model in Brief

The TFM is a formal mathematical model. Its main purpose is to facilitate understanding and analysis of the functionality of systems of any type – business, software, biological, mechanical, and so on [19]. The TFM represents the modeled functionality as a digraph (X, Θ), where X is a set of inner functional characteristics (further called functional features) of the system, and Θ is a topology set on these characteristics in a form of a set of cause-and-effect relations. Topological functioning models are comparable just like any digraphs. This property may be used to analyze similarities and differences among TFMs using a continuous mapping mechanism [20]. Since the 1990s the TFM is being elaborated for software development [21], at the beginning within the object-oriented paradigm and, later, within the model-driven development.

The TFM is characterized by the topological and functioning properties [22]. The topological properties are connectedness, neighborhood, closure, and continuous mapping. The functioning properties are cause-and-effect relations, cycle structure, inputs, and outputs. The composition of the TFM is presented in detail in [19]. In brief, it could be manual with the starting point in informal textual descriptions as within TFM4MDA (TFM for Model Driven Architecture) explained in multiple sources [23]–[25] and semi-automated with the starting point in use case scenarios as in the IDM toolset [26].

The main TFM construct is a functional feature (FF_i) that represents system's functional characteristic, e.g., a business process, a task, an action, or an activity [22]. It can be specified by a unique tuple (1).

$$FF_i = \text{<A, } \mathbf{R}\text{, } \mathbf{O}\text{, } \mathbf{PrCond}\text{, } \mathbf{PostCond}\text{, } \mathbf{Pr}\text{, } \mathbf{Ex}\text{>, where } FF_i \in \boldsymbol{FF} \quad (1)$$

Where tuple elements are as follows [19]:

- A is an object's action,
- **R** is a set of results of the object's action (it is an optional element),
- **O** is an object that gets the result of the action or a set of objects that are used in this action,
- **PrCond** is a set of preconditions or atomic business rules,
- **PostCond** is a set of post-conditions or atomic business rules,
- **Pr** is a set of providers of the feature, i.e., entities (systems or sub-systems) which provide or suggest an action with a set of certain objects,
- **Ex** is a set of executors (direct performers) of the functional feature, i.e., a set of entities (systems or sub-systems) that enact a concrete action.

As was mentioned, TFM's functional features have a topology over them in the form of cause-and-effect relations. The cause-and-effect relations between functional features define the cause from which the triggering of the effect occurs. The formal definition of the cause-and-effect relations and their combinations are given in [27]. It states that a cause-and-effect relation is a binary relationship that links the cause functional feature to the effect functional feature. In fact, in many cases this relation indicates a control flow transition in the system. The cause-and-effect relations (and their combinations) may be joined by the logical operators, namely, *conjunction (AND), disjunction (OR), or exclusive disjunction (XOR)*. The logic of the combination of cause-and-effect relations denotes system behavior (e.g., decision making) and a flow of execution of system's functions (e.g., in parallel or sequentially).

The TFM can be manually, but following the precise rules, transformed into most used UML (Unified Modeling Language) diagram types: class diagrams, activity diagrams, use cases and their textual specifications [28]. Besides, it can be transformed into Topological UML's [29] diagrams such as Topological Class diagrams, Topological Use Case diagrams, Activity diagrams, State Chart diagrams, and Sequence and Communication diagrams [30]. Thus, the

TFM holds all essential knowledge from the system's domain that should be implemented in design models and source code.

3.2 A Formal Reference Model

A topological functioning model represents a certain system's functionality formally. The question can be what the word "system" means in the topological functioning modeling. There are several definitions in the Oxford Learner's Dictionary of the meaning of "a system". For instance, a system is "a group of things, pieces of equipment, etc. that are connected or work together", or "an organized set of ideas or theories or a particular way of doing something"[†]. Both definitions are true for topological functioning modeling. This means that the TFM can be used for modeling any system that is represented by a group of things that are connected to work together according to an organized set of rules. That means that the TFM can specify a business system, an information system of this business system, a software system of the information system and so on – systems and their sub-systems. On the other hand, the TFMs (as mentioned in section 3.1) can be compared for similarities and differences by using continuous mapping of topological spaces. This means that the mapping can be used to analyze the changes in case of introducing new functions or modifying the already existing ones.

Therefore, if we consider distinguishing two domains – a problem and a solution, then we can speak about two systems – a system "AS IS" and a system "TO BE", correspondingly. In the case of topological modeling of functioning, if the TFM "AS IS" represents an existing system, then the TFM "TO BE" is modified TFM "AS IS" obtained as a result of mapping from the requirements to the system "TO BE" onto the TFM "AS IS".

Functional requirements within the TFM. Mappings from functional requirements (FRs) onto the TFM functional features can be one-to-zero, one-to-one, one-to-many, many-to-many, many-to-one, and zero-to-one [2], [3], [31]–[33]:

- *One-to-one* means that the functional requirement completely specifies one existing functional feature of the domain, for instance, the authorization of a registered user.
- *One-to-many, many-to-one,* and *many-to-many* cases relate to situations when specifications of functional requirements and/or functional features are too decomposed. One-to-many and many-to-one are special cases of the relation type "many-to-many". These cases can be caused by different levels of details in mapped elements. Such cases indicate and help in discovering decomposed, overlapping, or incomplete requirements.
- *One-to-zero* and *zero-to-one.* These are cases of new / undefined /missed functionality either in the specification of requirements, or in the model of the system. The "one-to-zero" occurs when one functional requirement describes new (or undefined) functionality of the system that can cause modification of the system and its TFM. The "zero-to-one" occurs when the requirements specification does not contain any functional requirement corresponding to the already defined functional characteristics. This can indicate the functionality that either will not be implemented in the "target" system, is new, or is missed. The new functionality will require changes in the existing processes of the system. The missed functionality either is not mentioned in the requirements specification or will be changed but it is not explicitly expressed.

As a result, mappings allow finding incomplete, additional, conflicting, unnecessary, as well as redundant functional requirements for the system functionality.

At present, a functional requirement is specified as a tuple (2) of its identifier id_{FR} and textual description $descr_{FR}$. The textual description is not limited to its format, it could be modified if necessary.

$$FR_i = < id_{FRi}, descr_{FRi} >, \text{where } FR_i \in \boldsymbol{FR} \quad (2)$$

[†] https://www.oxfordlearnersdictionaries.com/definition/english/system?q=systemType

Then a mapping from a set of functional requirements to a set of functional features can be specified as a tuple (3), where properties of this mapping can also be indicated as Boolean variables *isComplete* for indicating completeness and *isOverlapping* for indicating overlaps [2], [3].

$$FR2FF = < \boldsymbol{FR}, \boldsymbol{FF}, isComplete, isOverlapping> \quad (3)$$

Non-functional requirements within the TFM. Non-functional requirements (NFRs) can be mapped onto the TFM functional feature or a set of features by providing referencing in a way similar to the specification of the corresponding FRs. The possible types of NFRs mappings onto the TFM are one-to-zero, one-to-one, one-to-many, many-to-many, many-to-one, and zero-to-one [2], [3]:

- *One-to-one* is when one non-functional requirement is related to the concrete functional feature and must be implemented in the corresponding entities. For instance, a functional feature specifies retrieving of some set of records for some period from the database and a non-functional feature specifies that the accomplishment of the request must not exceed 3 milliseconds.
- *One-to-many* is when one non-functional requirement is related to all noted functional features and must be implemented in all the corresponding entities. For instance, several functional features specify retrieving data from the database and some successive calculations, and a non-functional feature that specifies that accomplishment of the request to the database must not exceed 3 milliseconds.
- *Many-to-one* is when more than one non-functional requirements are related to one noted functional feature and must be implemented in the corresponding entities. It could be considered a special case of the many-to-many relationship. For instance, two non-functional features specify the requirement for the language of the user interface and the requirement for the provided software interface. Both must be implemented in the input functional feature that specifies interaction with the users of the software.
- *One to zero*. One non-functional requirement is not related to any functional feature and is not traceable in the model and the code. This indicates that this requirement is out of the scope of the model and, hence, out of the scope of the system planned. There could be two causes, i.e., either the requirement is not appropriate, or the model lacks the required functionality. The latter may indicate an incomplete analysis of the required functions that are new for the system where the software will run.
- *Zero to one*. A functional feature is not related to any non-functional requirement. It is a reason to recheck the non-functional requirements.

As a result, mappings allow extending the specification of functional characteristics of the system with non-functional attributes and find incomplete, additional, conflicting, and redundant requirements to the system.

A non-functional requirement is specified as a tuple (4) of its identifier id_{NFR} and textual description $descr_{NFR}$, a dynamic characteristics D_{NFR} that can be expressed as a value or as a function (e.g. $D=f(p)$, where p is a parameter set of some function f), and a scope SC_{NFRi} of non-functional requirements. The scope may be a process, persistent data, or a whole system. The textual description is not limited to its format, it could be modified if necessary.

$$NFR_i = < id_{NFRi}, descr_{NFRi}, D_{NFRi}, SC_{NFRi} >, \text{where } NFR_i \in \boldsymbol{NFR} \quad (4)$$

Then a mapping from a set of non-functional requirements to a set of functional features can be specified as a tuple (5) for the general case *many-to-many* [2], [3].

$$NFR2FF = < \boldsymbol{NFR}, \boldsymbol{FF}> \quad (5)$$

Security Requirements within the TFM. As mentioned before, security requirements are considered a part of a set of non-functional requirements. However, security requirements are

more complex since they cannot be expressed positively like, for instance, functional requirements. They represent negative conditions, actions, and scenarios. Therefore, their integrations with the TFM cannot be expressed using the tuple (4) of a non-functional requirement. This tuple must be extended to represent information that is very important for the understanding its origin, and essence as well as for further analysis.

Therefore, the initial non-functional requirement's tuple should be modified as shown in expression (6). The dynamic characteristics that are important for performance requirements can be excluded. The following elements are added to the specification of a security requirement SR_i: a requirement's identifier id_{SRi}, a requirements textual explanation $descr_{SRi}$, a scope SC_{Sri} of the requirement (the list of a process, a persistent data, or a whole system can be extended with other elements), a set ***As*** of information assets that should be protected, a set ***Mrs*** of measures that should be applied for assets protection, a set ***Srs*** of sources of this requirement, a threat *thr*, a textual description of consequences *cnsq*, an assessment of the associated *risks*, and the kind of this security requirement according to the predefined list – *kind*.

$$SR_i = < id_{SRi}, descr_{SRi}, SC_{Sri}, \textbf{As, Mrs, Srs,} thr, cnsq, risk, kind >, \text{where } SR_i \in \boldsymbol{NFR} \quad (6)$$

A mapping from a set of security requirements to a set of functional features is specified as the same tuple (5) for the general case *many-to-many* of non-functional requirements since security requirements are a subset of non-functional requirements. Mappings from the security requirements onto the TFM have the following meaning:

- *One-to-one* is when a security requirement is linked to a certain functional feature and must be implemented in the corresponding design entities. For instance, when a certain validation of input data must be implemented before transferring data to processing.
- *One-to-many* is when one security requirement is linked to all noted functional features and must be implemented in all the corresponding entities. For instance, when transferred data must be encrypted before transferring to the database.
- *Many-to-one* as a special case of the *many-to-many* relationship is when more than one security requirement is linked to one noted functional feature and must be implemented in the corresponding entities. For instance, when validation of input data before and encryption of output data after must be implemented in the same function.
- *One to zero*. A security requirement is not linked to any functional feature. This indicates that a model lacks its implementation. This case may indicate an incomplete analysis of the required functions of the system or some new functionality that is not but must be implemented.
- *Zero to one*. A functional feature is not linked to any security requirement. This could be a correct case, but for input and output functional features this case must be re-checked.

4 Illustrative Example

Let us consider the example of the TFM for a library system. The TFM (Figure 1 and Figure 2) specifies the main functionality provided by the library, i.e. registering persons as readers, giving out and taking back the books as well as imposing a fine in case of damages to the book or the exceeded loan time.

Let us assume that the task is to create new software that should support librarians' work. The new software must implement three secure design principles – input data must be validated before processing (P1), output data must be checked for not breaking confidentiality (P2), and *a password must be encrypted before saving* (P3).

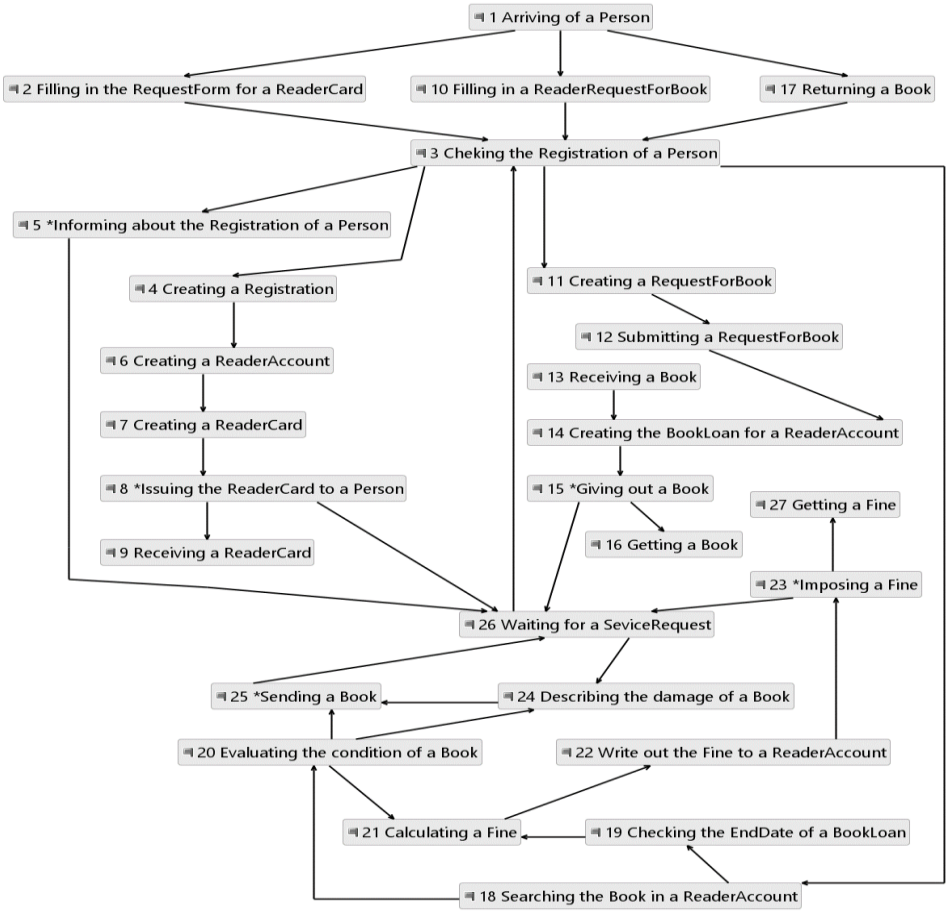

Figure 1. The topological functioning model (simplified) of the library operation [2]

id	Description	Action	Result	Object	S	Ex
1	Arriving of a Person	arrive	Person	Person	E	P
2	Filling in the RequestForm for a ReaderCard	request	RequestForm	ReaderCard	I	P
3	Cheking the Registration of a Person	check	Registration	Person	I	L
4	Creating a Registration	create	Registration	Registration	I	L
5	*Informing about the Registration of a Person	inform	Registration	Person	I	L
6	Creating a ReaderAccount	create	ReaderAccount	ReaderAccount	I	L
7	Creating a ReaderCard	create	ReaderCard	ReaderCard	I	L
8	*Issuing the ReaderCard to a Person	issue	ReaderCard	ReaderCard	I	L
9	Receiving a ReaderCard	receive	ReaderCard	ReaderCard	E	R
10	Filling in a ReaderRequestForBook	fillIn	ReaderRequestForBook	ReaderRequestForBook	I	L
11	Creating a RequestForBook	create	RequestForBook	RequestForBook	I	R
12	Submitting a RequestForBook	submit	RequestForBook	RequestForBook	I	L
13	Receiving a Book	receive	Book	Book	I	L
14	Creating the BookLoan for a ReaderAccount	checkOut	BookLoan	Book	I	L
15	*Giving out a Book	giveOut	Book	Book	I	L
16	Getting a Book	get	Book	Book	E	R
17	Returning a Book	return	Book	Book	E	R
18	Searching the Book in a ReaderAccount	search	Book	ReaderAccount	I	L
19	Checking the EndDate of a BookLoan	check	EndDate	BookLoan	I	L
20	Evaluating the condition of a Book	evaluate	condition	Book	I	L
21	Calculating a Fine	calculate	Fine	Fine	I	L
22	Write out the Fine to a ReaderAccount	writeOut	Fine	ReaderAccount	I	L
23	*Imposing a Fine	impose	Fine	Fine	I	L
24	Describing the damage of a Book	describe	damage	Book	I	L
25	*Sending a Book	send	Book	Book	I	L
26	Waiting for a SeviceRequest	wait	ServiceRequest	ServiceRequest	I	L
27	Getting a Fine	get	Fine	Fine	E	R

Figure 2. The specification of TFM functional features, where S – subordination, I – inner of the system, E – external to the system, Ex – the executor, R – the reader, L – the librarian, P – the person [2]

Analysis of the TFM indicates that it has one input vertex – functional feature 1 – that specifies an event of arriving of a person that will start interaction with the system. Additionally, we can see that the TFM has four output functional features (marked with "*") – 5 "Informing about the Registration of a Person", 8 "Issuing the ReaderCard to a Person", 23 "Imposing a fine", and 25 "Sending a Book".

Analysis of the domain objects that are used in the system shown that several of them may contain confidential personal data (*protectable information assets*). They are a ReaderCard, a Registration, a ReaderAccount, a Person and a Fine.

According to principles P1 and P2, the analysis of input and output functionality must be done. First, we must check whether these domain objects are used in the input and output functional features. Functional features 1, 5, and 8 include the object *Person*. However, functional feature 1 represents an external functionality and is out of the scope. Functional feature 5 includes also the object *Registration*. Functional feature 8 includes the object *ReaderCard*. Functional feature 23 includes the object *Fine*. Analysis of other functional features shows that there is a function for checking the authentication of the visitor (feature 3) where both objects *Registration* and *Person* are included.

According to principle P3, an analysis of the functionality that saves the password must be done. Looking at the TFM, it is visible that the password is saved as a part of the created *ReaderAccount* in functional feature 6.

Summarizing the result of this small analysis the following mappings from the required secure design principles to functional features can be specified:

- *One-to-many: P2* to functional features 5, 8, and 23.
- *One-to-one: P1* to functional feature 3; and *P3* to functional feature 6.

Thus, one can specify the corresponding security requirements (Figure 3) and map them to the functional features defined (Figure 4). *SR1* comes from the secure design policy represented by *P1*, *SR2* – from the secure design policy represented by *P2* as well as General Data Protection Regulation (GDPR), and *SR3* – from the secure design policy represented by *P3* and GDPR. Correspondingly, *SR2* is referred to the set of functional features {5, 8, 3}, *SR1* to functional feature 3, and *SR3* to functional feature 6.

id	descr	SC	As	Mrs	Src	thr	cnsq	risk	kind
SR1	Input data must be validated before processing	Process	Registration, Person	Validation procedure	Secure design policy	inserting malicious data	can be start of killchain	medium	attack
SR2	Output data must be checked for not breaking the confidentiality	Process	Registration, ReaderCard, Fine	Filtering	Secure design policy, GDPR	exposing confidential data	leak of personal data	medium	privacy
SR3	A password must be encrypted before saving	Persistent data	ReaderAccount	Encryption	Secure design policy, GDPR	exposing confidential data	leak of personal data	medium	privacy

Figure 3. Security Requirements according to the defined principles

NRFs		FF	
id	descr	id	descr
SR2	Output data must be checked for not breaking the confidentiality	5	Informing about the Registration of a Person
SR2	Output data must be checked for not breaking the confidentiality	8	Issuing the ReaderCard to a Person
SR2	Output data must be checked for not breaking the confidentiality	23	Imposing a Fine
SR1	Input data must be validated before processing	3	Checking the Registration of a Reader
SR3	A password must be encrypted before saving	6	Creating a ReaderAccount

Figure 4. Mappings from security requirements to functional features

Although security requirements have properties different from other non-functional requirements, they still can be mapped onto the TFM functional features. Thus, TFM as a reference model allows the showing of required functionality and its non/extra-functional characteristics including security requirements already at the stage of domain modeling and analysis. Besides, the analysis of needed security requirements may also include an assessment of risks.

5 Conclusion

Security requirements concern the provision of confidentiality, integrity, availability, and privacy of information assets. In this article the concept of a formal reference model, the topological functioning model, is presented. The topological functioning model is a mathematical model that can be used for referencing functional and non-functional requirements including security requirements. Besides, its formal nature and holistic representation of a domain suggest using formal analytical means for security requirements analysis and specification. Thanks to the continuous mapping between graphs (that can be called trace models), dependencies among existing processes and their implementation in analytical, design and code elements are explicit.

Formal tracing of security requirements onto the TFM functional features allows tracing them forward to design and code constructs and backward to the sources of origin, discovering possible incompleteness and conflicts in software requirements during the problem analysis and design activities. Additionally, the TFM can be used as an analytical means for discovering dependent or affecting functionality and functionality limitations.

Implementation of the presented approach and integration of it with the existing toolset is the future research direction. Besides, perspectives of using TFM for modeling malicious behavior also should be investigated.

References

[1] L. Liu, “Security and Privacy Requirements Engineering Revisited in the Big Data Era,” in *2016 IEEE 24th International Requirements Engineering Conference Workshops (REW)*, Sep. 2016, pp. 55–55. Available: https://doi.org/10.1109/REW.2016.023

[2] E. Nazaruka and J. Osis, “The Topological Functioning Model as a Reference Model for Software Functional and Non-functional Requirements,” in *13th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2018)*, vol. 1, 2018, pp. 467–477. Available: https://doi.org/10.5220/0006811204670477

[3] E. Nazaruka and J. Osis, “The Formal Reference Model for Software Requirements,” in *Evaluation of Novel Approaches to Software Engineering. ENASE 2018. Communications in Computer and Information Science*, vol. 1023, E. Damiani, G. Spanoudakis, and L. Maciaszek, Eds. Springer, 2019, pp. 352–372. Available: https://doi.org/10.1007/978-3-030-22559-9_16

[4] S. Turpe, “The Trouble with Security Requirements,” in *2017 IEEE 25th International Requirements Engineering Conference (RE)*, Sep. 2017, pp. 122–133. Available: https://doi.org/10.1109/RE.2017.13

[5] I. Nigmatullin, A. Sadovykh, N. Messe, S. Ebersold, and J.-M. Bruel, “RQCODE – Towards Object-Oriented Requirements in the Software Security Domain,” in *2022 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, Apr. 2022, pp. 2–6. Available: https://doi.org/10.1109/ICSTW55395.2022.00015

[6] Q. Rouland, B. Hamid, J.-P. Bodeveix, and M. Filali, “A Formal Methods Approach to Security Requirements Specification and Verification,” in *2019 24th International Conference on Engineering of Complex Computer Systems (ICECCS)*, Nov. 2019, pp. 236–241. Available: https://doi.org/10.1109/ICECCS.2019.00033

[7] D. Mellado, E. Fernández-Medina, and M. Piattini, “Applying a Security Requirements Engineering Process,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence*

and Lecture Notes in Bioinformatics), 2006, vol. 4189 LNCS, pp. 192–206. Available: https://doi.org/10.1007/11863908_13

[8] L. Yin and F.-L. Qiu, "A novel method of security requirements development integrated common criteria," in *2010 International Conference On Computer Design and Applications*, Jun. 2010, vol. 5, pp. V5–531–V5–535. Available: https://doi.org/10.1109/ICCDA.2010.5541109

[9] I. A. Tøndel, J. Jensen, and L. Røstad, "Combining Misuse Cases with Attack Trees and Security Activity Models," in *2010 International Conference on Availability, Reliability and Security*, Feb. 2010, pp. 438–445. Available: https://doi.org/10.1109/ARES.2010.101

[10] A. Nhlabatsi *et al.*, "Managing Security Control Assumptions Using Causal Traceability," in *2015 IEEE/ACM 8th International Symposium on Software and Systems Traceability*, May 2015, pp. 43–49. Available: https://doi.org/10.1109/SST.2015.14

[11] Z. Zhioua, Y. Roudier, and R. B. Ameur, "Formal Specification and Verification of Security Guidelines," in *2017 IEEE 22nd Pacific Rim International Symposium on Dependable Computing (PRDC)*, Jan. 2017, pp. 267–273. Available: https://doi.org/10.1109/PRDC.2017.51

[12] Q. Ramadan, M. Salnitriy, D. Struber, J. Jurjens, and P. Giorgini, "From Secure Business Process Modeling to Design-Level Security Verification," in *2017 ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, Sep. 2017, pp. 123–133. Available: https://doi.org/10.1109/MODELS.2017.10

[13] C. Ponsard, J.-C. Deprez, and R. Darimont, "Formalizing Security and Safety Requirements by Mapping Attack-Fault Trees on Obstacle Models with Constraint Programming Semantics," in *2020 IEEE Workshop on Formal Requirements (FORMREQ)*, Aug. 2020, pp. 8–13. Available: https://doi.org/10.1109/FORMREQ51202.2020.00009

[14] M. Quamara, G. Pedroza, and B. Hamid, "Multi-layered Model-based Design Approach towards System Safety and Security Co-engineering," in *Companion Proceedings - 24th International Conference on Model-Driven Engineering Languages and Systems, MODELS-C 2021*, 2021, pp. 274–283. Available: https://doi.org/10.1109/MODELS-C53483.2021.00048

[15] D. Tsoukalas, M. Siavvas, M. Mathioudaki, and D. Kehagias, "An Ontology-based Approach for Automatic Specification, Verification, and Validation of Software Security Requirements: Preliminary Results," in *2021 IEEE 21st International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, Dec. 2021, pp. 83–91. Available: https://doi.org/10.1109/QRS-C55045.2021.00022

[16] J. J. Olthuis, R. Jordão, F. Robino, and S. Borrami, "VrFy: Verification of Formal Requirements using Generic Traces," in *2021 IEEE 21st International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, Dec. 2021, pp. 177–183. Available: https://doi.org/10.1109/QRS-C55045.2021.00034

[17] W. Wang, F. Dumont, N. Niu, and G. Horton, "Detecting Software Security Vulnerabilities Via Requirements Dependency Analysis," *IEEE Transactions on Software Engineering*, vol. 48, no. 5, pp. 1665–1675, May 2022, Available: https://doi.org/10.1109/TSE.2020.3030745

[18] C. Schmitt and P. Liggesmeyer, "Instantiating a model for structuring and reusing security requirements sources," in *2015 IEEE 2nd Workshop on Evolving Security and Privacy Requirements Engineering (ESPRE)*, Aug. 2015, pp. 25–30. Available: https://doi.org/10.1109/ESPRE.2015.7330164

[19] J. Osis and E. Asnina, "Topological Modeling for Model-Driven Domain Analysis and Software Development : Functions and Architectures," in *Model-Driven Domain Analysis and Software Development: Architectures and Functions*, Hershey, PA: IGI Global, 2011, pp. 15–39. Available: https://doi.org/10.4018/978-1-61692-874-2.ch002

[20] E. Asnina and J. Osis, "Computation Independent Models: Bridging Problem and Solution Domains," in *Proceedings of the 2nd International Workshop on Model-Driven Architecture and Modeling Theory-Driven Development*, 2010, pp. 23–32. Available: https://doi.org/10.5220/0003043200230032

[21] J. Osis, E. Asnina, and A. Grave, "Computation Independent Representation of the Problem Domain in MDA," *e-Informatica Software Engineering Journal*, vol. 2, no. 1, pp. 29–46, 2008.

[22] J. Osis and E. Asnina, "Is Modeling a Treatment for the Weakness of Software Engineering?," in *Model-Driven Domain Analysis and Software Development*, Hershey, PA: IGI Global, 2011, pp. 1–14. Available: https://doi.org/10.4018/978-1-61692-874-2.ch001

[23] E. Asnina, “The Computation Independent Viewpoint: a Formal Method of Topological Functioning Model Constructing,” *Applied Computer Systems*, vol. 26, pp. 21–32, 2006.

[24] J. Osis, E. Asnina, and A. Grave, “MDA oriented computation independent modeling of the problem domain,” in *Proceedings of the 2nd International Conference on Evaluation of Novel Approaches to Software Engineering – ENASE 2007*, 2007, pp. 66–71. Available: https://doi.org/10.5220/0002584500660071

[25] J. Osis, E. Asnina, and A. Grave, “Formal Problem Domain Modeling within MDA,” in *Software and Data Technologies: Second International Conference, ICSOFT/ENASE 2007, Barcelona, Spain, July 22-25, 2007, Revised Selected Papers*, J. Filipe, B. Shishkov, M. Helfert, and L. A. Maciaszek, Eds. Berlin, Heidelberg: Springer, 2008, pp. 387–398. Available: https://doi.org/10.1007/978-3-540-88655-6_29

[26] A. Šlihte and J. Osis, “The Integrated Domain Modeling: A Case Study,” in *Databases and Information Systems: Proceedings of the 11th International Baltic Conference (DB&IS 2014)*, 2014, pp. 465–470.

[27] E. Asnina and V. Ovchinnikova, “Specification of decision-making and control flow branching in Topological Functioning Models of systems,” in *Proceedings of the 10th International Conference on Evaluation of Novel Approaches to Software Engineering – MDI4SE*, pp. 364–373, 2015. Available: https://doi.org/10.5220/0005479903640373

[28] J. Osis and E. Asnina, “Derivation of Use Cases from the Topological Computation Independent Business Model,” in *Model-Driven Domain Analysis and Software Development*, Hershey, PA: IGI Global, 2011, pp. 65–89. Available: https://doi.org/10.4018/978-1-61692-874-2.ch004

[29] U. Donins, J. Osis, A. Slihte, E. Asnina, and B. Gulbis, “Towards the refinement of topological class diagram as a platform independent model,” in *Proceedings of the 3rd International Workshop on Model-Driven Architecture and Modeling-Driven Software Development, MDA and MDSD 2011, in Conjunction with ENASE 2011*, 2011, pp. 79–88.

[30] J. Osis and U. Donins, “Formalization of the UML Class Diagrams,” in *Evaluation of Novel Approaches to Software Engineering*, New York: Springer, Berlin, Heidelberg, 2010, pp. 180–192. Available: https://doi.org/10.1007/978-3-642-14819-4_13

[31] J. Osis and E. Asnina, “Enterprise Modeling for Information System Development within MDA,” in *Proceedings of the 41st Annual Hawaii International Conference on System Sciences (HICSS 2008)*, Jan. 2008, pp. 490–490. Available: https://doi.org/10.1109/HICSS.2008.150

[32] J. Osis and E. Asnina, “A Business Model to Make Software Development Less Intuitive,” in *2008 International Conference on Computational Intelligence for Modelling Control & Automation*, 2008, pp. 1240–1245. Available: https://doi.org/10.1109/CIMCA.2008.52

[33] E. Asnina, B. Gulbis, J. Osis, G. Alksnis, U. Donins, and A. Slihte, “Backward requirements traceability within the topology-based model driven software development,” in *Proceedings of the 3rd International Workshop on Model-Driven Architecture and Modeling-Driven Software Development, MDA and MDSD 2011, in conjunction with ENASE 2011*, 2011, pp. 36–45.