STUDIA INFORMATICA

Formerly: *Zeszyty Naukowe Politechniki Śląskiej, seria INFORMATYKA*
Quarterly

Volume 29, Number 2B (77)

Andrzej BIAŁAS

SEMIFORMAL COMMON CRITERIA COMPLIANT IT SECURITY DEVELOPMENT FRAMEWORK

Silesian University of Technology Press
Gliwice 2008

STUDIA INFORMATICA **Volume 29, Number 2B (77)**

Formerly: *Zeszyty Naukowe Politechniki Śląskiej, seria INFORMATYKA* Nr kol. 1784

STUDIA INFORMATICA is indexed in INSPEC/IEE (London, United Kingdom)

CONTENTS

SPIS TREŚCI

1. Introduction and motivation

Informatics is a discipline of science and technology that focuses on the processing, storing and transferring of information. This monograph deals with the selected yet important aspects of the foundations which enable these activities.

The development of e-business, e-government or e-health applications and the critical information infrastructure protection strongly depends on the development of trust and confidence technologies. The development of these technologies, however, needs an assurance basis.

Assurance is the confidence that an entity, i.e. IT (ICT) (*Information (and Communication) Technology*) product or system, called the TOE (*Target of Evaluation*), meets the security objectives which are specified for it. The Common Criteria standard (abbrev. CC), i.e. ISO/IEC 15408, is a well established methodology dealing with the creation of assurance. This standard is currently available in version 3.1 (from September 2006) and provides criteria for evaluating IT products or systems with focus on their assurance. The importance of the Common Criteria methodology is still growing. The basic information, latest standard versions and related documentation are available in the Common Criteria Portal [42].

The monograph presents the Common Criteria- and UML-based (*Unified Modelling Language*) method for IT security-related products development, ensuring the expected assurance level for them. The method represented by the developed UML framework is generally compliant with the Common Criteria, but it introduces some significant extensions and improvements as well. It can be used directly as the development method or can be the basis of a computer-aided tool that supports design and evaluation processes.

The monograph deals with the following main issues that should be briefly presented at the beginning:

- understanding the Common Criteria philosophy concerning the assurance,
- general aspects of the semiformal, UML-based modelling in the information security domain.

1.1. Assurance from the Common Criteria point of view

Assurance can be considered as a passport of the given IT product or system to the security critical applications. The basis of the Common Criteria philosophy is the assumption that the assurance foundation is created during a rigorous IT development process. The assurance verification, on the other hand, is carried out during independent evaluation and later during operation of a certified IT product or system. The users, who entrust their information and other business assets to IT systems, should have confidence that these systems with their safeguards guarantee the right assurance, i.e. they offer intended functionality and eliminate unexpected, malicious behaviour.

Almost all today's systems are security-critical and should be developed in a special way. It is difficult, mainly because of the conflict between the cost of the development and verifiable correctness. The products or systems security should be considered at their early development stage and within development context. Security mechanisms should not be inserted into products or systems blindly, but prudently, and this requires security engineering knowledge, methods and tools. It should be noted that the IT security efficiency seems to be higher than simpler methods and techniques that are used to achieve it. Modelling methods serve this purpose very well.

The monograph focuses on the development method of IT products or systems providing the right assurance. IT consumers who use different IT products or systems expect that these products should work just as they were designed. The consumers also require that the built-in functionality of these products or systems should provide well defined, expected behaviour and eliminate unexpected behaviour. In other words, the IT consumers need assurance for their products and systems. The above definition of assurance implies two basic issues:

- precise and coherent specification of the security objectives for the TOE provided by the IT security development process,
- creating the basis of the above mentioned confidence (assurance) provided by the independent evaluation and certification by independent bodies.

These two goals can be achieved by the Common Criteria (ISO/IEC 15408) methods [38-40], [45-46], [60].

IT products or systems should be developed in a rigorous manner. Rigorous is understood as more precise, more coherent, mathematically based, supported by the rationale and verification processes, etc. In order to achieve this, a more formalized IT security development methodology should be applied. The formalization, however, has its own limitations (e.g. cost, special trainings for developers) and can be applied in a reasonable way only for these areas where it can bring real advantages.

The Common Criteria IT security development methodology has precise, but rather informal character [60], with the semiformal functional [39] and assurance [40] components. These components function as the specification language, but only at the security requirements elaboration stage. It should be noted that “semiformal” means: “expressed in a restricted syntax language with defined semantics” [38].

The CC standard can be considered as an object-oriented semiformal modelling language or toolset for a specific context (e.g. IT security) [51]. Common Criteria impose rigorous development and evaluation on any security-related product, which depends mostly on the declared Evaluation Assurance Level (in the range: EAL1-EAL7, where the EAL7 is the highest value). Basically, stricter discipline in development and evaluation means better assurance. The development and evaluation processes are very complicated due to many details, dependencies and feedbacks, which should be taken into consideration, and rather difficult rationales. That is why the need of specialized frameworks and computer-aided tools that are based on them is important and growing.

All secured IT hardware or software products or systems, i.e. TOEs, should be rigorously developed and then evaluated. The TOEs are created on the basis of the security requirements specifications, well defined within the standard, i.e.:

- Security Target (ST) – an implementation-dependent set of security requirements for a given TOE, satisfying specific consumer needs,
- Protection Profile (PP) – an implementation-independent set of security requirements for a group of similar TOEs, meeting consumer needs.

Before developing the TOE, an ST or PP document should be created. The TOE may be developed:

- straight on the ST basis; the evaluator first evaluates the ST documents and then the corresponding TOE, according to its specification;
- on the ST created according to the evaluated PP; with the use of positively evaluated Protection Profiles, specific Security Targets describing TOEs are developed and evaluated in two stages: first the ST is evaluated (PP compliance) and then the TOE.

1.2. Selected aspects of the semiformal, UML-based modelling

The methods dealing with the UML [36], [49], [76], [102] are very promising, also in the information security domain. They prove a unified approach to the products and their security features description. The UML creates unprecedented possibilities for secure-critical products or systems development which are feasible in industrial context. The UML is de facto an industrial modelling standard now, supported by different analyses, testing, simulations, and transformation tools. The UML community is still growing thus facilitating the use of the CC methodology is very important, especially for the increasing number of the secure COTS (commercial off-the-shelf) products.

The word framework has a more general meaning there as it was assumed in the UML: "a framework" is an architectural pattern that provides an extensible template for applications within a domain. This template specifies a set of mechanisms that define a skeleton of an architecture together with the "slots, tabs, knobs, and dials" that are exposed to adapt the framework to a specific context" [36].

The modelling approach is growing and the UML-approach is commonly used by IT developers to solve design problems. The UML models can be simplified, presenting the selected issues of the entire concepts, and not always are they transferred to the executable code. The IT developers face IT security problems during their work too. It will be easier for them to use the same approach to solve both issues. For these reasons it is vital to integrate the IT product design and IT security development. This can be achieved by using the common approach – the UML modelling approach. Aspect-oriented modelling [4] extension can be helpful to express the complex, cross-cutting concerns.

The current state of researches, presented later, shows that modelling is one of mainstream tools for the information systems industry but not for the security specialty within the industry. IT security modelling requires a slightly different approach in comparison with the IT systems modelling, which is mainly focused on the functionality offered to its users. IT security must be considered as a whole, but it consists of many security items dealing with different IT items, like: properties, functions, modules and mechanisms with cross-cutting concerns inside.

Security-related products design, as other IT systems design, needs several decompositions (top-down analyses) and compositions (bottom-up syntheses), alternating each other, and the separation of concern is a well known general principle. The composition/decomposition problems of an IT system with considered IT security are not trivial. It is very difficult for the designers to comprehend a large number of functional elements with their numerous details on one side, and global behaviour and properties of the system connected with them on the other. Additionally, the level of abstraction used in the IT domain and in the IT security domain may be inconsistent. The modelling can help to solve these problems, contributing to simplify and better organize the IT security development process. The modelling is very promising but there is still a challenge to build more effective models with the use of object-oriented methods.

Any IT design can be expressed by its vertical (functionality, structure, etc.) and horizontal (security concerns, objectives, requirements, etc.) views. For both, a unified approach based on the CC and UML is proposed, open to new achievements and the changes occurring in standards in these two areas.

1.3. Objectives of the work and the monograph contents

Generally, the issue presented there concerns how to perform the IT security development rigorously, i.e. more precisely, formally, and consistently, and more effectively, i.e. more quickly and cheaply.

In this monograph, on the basis of:

- the analysis of the assurance concept provided by the Common Criteria methodology (Section 1.1),
- the analysis of the modelling issues concerning the information security domain, focusing mainly on the security engineering (Section 1.2, Section 2.1),
- the analysis of the Common Criteria methodology and the identification of the developers' needs concerning the support they require in most difficult and complicated tasks performed during the IT security development process (Section 2.2),

it will be shown that it is possible to:

- elaborate the IT Security Development Framework (ITSDF) and the related method, specifying step by step the developers' efforts focused on the IT product or system development at the required assurance level, particularly:
  - elaborate a semiformal model of the IT security development process with the use of the UML approach,
  - create a set of semiformal and formal means and methods to build IT security related specifications of the TOE in a more precise and consistent way,
- implement this framework as the computer-aided tool facilitating these efforts.

The main concept assumes concurrent development of three UML-based models, representing the ITSDF framework (Section 2.3). The first one, the model of the security-related product, presents its elements, functionality offered to users, requirements and working environment. On this basis the second model – the security model of the security-related product – is created, using the Common Criteria method of specification. The third one is responsible for evaluation of the developed security model.

To provide the IT security developers with the Common Criteria compliant framework, and the tool based on it, the following methodology was applied – presented in this monograph and in other several publications of the author:

- understanding the Common Criteria philosophy and applications,
- identifying the CC developers needs,
- identifying the gaps of existing solutions,

- developing a general concept of the ITSDF framework,
- building its model incrementally and recursively,
- model validation during case studies with experts, trainings and discussions,
- implementing the model as the computer-aided tool,
- publication and presentation of the concept and solutions at many scientific conferences or workshops, including the NATO Advanced Research Workshop in Gdansk [13], the World Congress in Applied Computing in Las Vegas [22], [29], 8th International Common Criteria Conference in Rome [28],
- model refinement and supplementing some formal issues,
- planning to continue the work, based on the results and current experiences.

The monograph consists of 13 chapters including this introductory one, showing full range of works and their results.

Chapter 2 starts from a short presentation of the current state of technology, focusing on the gaps identification. The review includes the most relevant publications concerning: IT security development process, general modelling aspects, UML and OCL (Object Constraint Language) implementation, semiformal and security engineering methods, particularly concerning Common Criteria, and computer aided tools. This chapter also summarizes the Common Criteria development process, identifying points whose support for the developers is especially required (i.e. developers' needs) and, generally, presents the concept of the developed framework (main structural model and the state machine representing its elaboration). Short discussion of the available specification means is added, that is the motivation to improve them.

Chapter 3 introduces "a specification language" implemented as a design library available for the IT security developers. The library consists of CC functional and assurance components and a set of generics – all represented as the UML classes.

Chapter 4 encompasses the initial stage of the development methodology, focusing on capturing the basic features of an IT product or system as "input data".

Chapters 5 through 10 are of the different sizes because they reflect the security target development stages. These stages are expressed by the use of the introduced ITSDF framework. First sections of each of these chapters summarize (on the UML diagrams) what the CC standard offers to developers, the successive sections represent the author's contribution, including formal aspects concerning a given development stage, placed in the final sections.

Chapter 11 concerns the evaluation model and processes that were fully implemented within the tool, but are not discussed in details in this monograph.

Chapter 12 presents the software implementation of the IT Security Development Framework, including the library of specification means. The presentation is very concise, exemplifying some solutions and referring to the technical documentation.

The final chapter, 13, summarizes the goals, range and results of the whole work.

The monograph includes also five appendices devoted to: the basic Common Criteria terminology, the selected OCL syntax and semantics used for the UML model representation in the monograph, selected cryptographic terms, naming of the chosen terms and an example of the ST for a firewall.

The work is compliant with the [60]. Please note some differences (rather simplifications) influenced by the latest CC v.3.1. [42].

It was assumed that the reader is familiar with the basic issues included in the Common Criteria standard and related documents, because it is not possible to discuss them in this monograph. Basic terminology contained in the Appendix A may be very helpful.

1.4. Permission and acknowledgement

The screenshots of the ITSDF-tool, developed on the basis of the author's concepts and under the author's supervision, are printed with their owner's permission, i.e. the Institute of Control Systems – Chorzów, Poland. The initial software tool version, called POZIT, was supported in part by a grant from the State Committee for Scientific Research (the Ministry of Science and Information Society Technologies).

The author[1] wishes to thank the President of the Management Board of the Institute of Innovations and Information Society (INSI), Mr Przemysław Gnitecki for his decision to finance the publication of this work. The author also wishes to thank for the support from INSI R&D Director, Mr Leszek Żychoń, and for the support from Ms Barbara Flisiuk, Ms Irena Styczeń, Mr Oleksandr Pidlisny, Mr Jacek Bagiński – key members of the ITSDF-tool development team.

[1] Andrzej BIAŁAS: Instytut Innowacji i Społeczeństwa Informacyjnego, ul. Wita Stwosza 7, 40-954 Katowice, Polska, Andrzej.Bialas@insi.pl

2. Concept of the IT Security Development Framework

This chapter presents the background (existing solutions, their gaps and the developers' needs with respect to the support of their activities) and the general concept of the created framework.

2.1. Current state of technology

The methods reviewed there do not deal with highly secure systems but systems in general, called "High integrity systems", where the cost of loss or failure is significant. This term, introduced in [47], covers also safety-critical systems (such as those controlling aircraft, nuclear power plants and so on), and systems, where failure may result in a significant financial loss or embarrassment, like launch control systems.

High integrity systems require specific, well formalized methods and tools used for their development. Some of them are UML-based. The formal methods are used rather as the extensions of the UML-based ones, to support them in specific areas.

The development process of high-integrity systems must be fully controlled, including the use of ready-made components, like library components, and should satisfy many constraints to avoid any ambiguities with respect to the specification and implementation.

The UML is a modelling language, intended to be a commonly shared language used not only in medium- to large-scale software projects, but also in very diverse areas of application, very often beyond computer science and telecommunications. IT security is one of these areas.

Currently, many IT security-related methods, presented below, refer to the UML, trying to adopt this modelling approach to solve IT security problems.

The review of existing works is focused on the publications and the tools dealing with:

- the use of the UML for IT security development, especially for the Common Criteria development and evaluation,
- formal and semiformal methods used for the IT security development, as the support or extension of the UML-based methods,
- IT security frameworks, especially UML-based,

- computer-aided tools supporting design and evaluation,
- some new trends in the information and electronic services security,

closely connected with the proposed framework.

2.1.1. Security engineering – the selected issues

To better understand the methodology presented in this monograph and its relationship with the security engineering, a short review of this domain is provided at the beginning.

The Common Criteria standard has been intended to provide a semiformal or formal way to specify and evaluate the security properties of hardware or software. It should be noted that most development and evaluation problems occur with software. A very important thing is low quality of the software, especially in the case of COTS. Many existing information security issues are implied [97] "by unfortunate trends in the way software is produced, acquired, deployed, and then used." There is a general need of the right use of software engineering methods concerning also security engineering principles.

Security engineering, like other engineering areas, can be considered both as an art and a science, encompassing and consistently applying a set of principles that are intended to ensure the consistency and quality of the results it produces [79]. There are some common engineering principles that can be interpreted and applied in security engineering – the engineers:

- take responsibility for the results they produce,
- have developed material goods for ages – a very old area of human activity,
- do not begin work until the requirements of the system are well understood and documented,
- address requirements as an ordered list, do not consider any of them separately,
- study the strengths and limitations of the materials they use,
- prefer simplicity,
- do not mix the controls intended for the exclusive use of the management with those intended for end users,
- prefer to reuse proven designs.

The problem is that "IT people in general and security people in particular do not consistently apply traditional engineering principles to their work" [79].

Security engineering is not software engineering, although there exist areas covered by both. Security engineering does not focus on how to build secure software, but rather on how to build secure systems using available software [79].

Security engineering is often concerned with safeguards, especially against viruses, direct network attacks or with cryptography applications, although it deals rather with how to integrate the intended product or system functionality with its security at an early development stage.

Security engineering emphasizes an analysis of the system requirements, for example around the following questions [53]:

- "What is the purpose of this system?
- What types of data will it store or process?
- How critical is the data or its accuracy?
- Does the security of the information warrant the cost?
- Where is the true threat likely to come from?"

Security engineering is an important element of the system designing process – its principles should be taken into consideration when a project is first conceived, during the development stages of a system, to ensure effective measures implementation, and for the security maintenance when the system is used. Security engineering encompasses also non-technical aspects – organizational, dealing with management, even with culture, tradition and education.

The key issue of security engineering can be the design for securability concept, understood as applying engineering principles to system security design in accordance with the fact that "no system can be designed to be secure, but can include the necessary prerequisites to be secured during operations" [54]. The design for securability encompasses very distinguished means and ways, like: mutual trust between system owners and operators, requirements engineering, system modelling, methods and tools for design support, risk management, etc.

Generally, two types of system models can be developed:

- design models – built before the system is implemented,
- analytical models – built for the existing systems.

System models are created on the basis of:

- system requirements,
- high-level description of the system,
- models of system components.

The system models are analyzed and then modified on the basis of the analysis results, using distinguished methods and tools (automatic or manual activities). During the modelling process the system description is transformed from a textual form, through a set of loose security-related statements, to the validated, consistent set of statements.

The Common Criteria standard supports different aspects of the design for securability approach at different levels, and they were taken into consideration too within the proposed

IT Security Development Framework. Three steps of this approach are usually considered [54] (compare with Fig. 2.1):

- capturing interactions and relations between the IT system and its environment,
- formulating a set of security requirements on the IT system,
- implementing this set of requirements.

The first issue is rather poorly supported by the CC, although its results decide about the effectiveness of the whole design. It depends on the IT security developers' knowledge, experience and used "high-level" methods. The proposed IT Security Development Framework recommends to use the UML approach, especially the UMLsec approach discussed later (see 2.1.2). The second issue is fully supported by the Common Criteria. The third, depending on the development environment for the software or hardware, is supported by different existing methods and tools, discussed later.

The key principle of security engineering, applicable especially to software engineering, is the *separation of concerns*. This principle says that software should be decomposed in such a way that different "concerns" or aspects of the problem at hand are solved in well separated modules or parts of the software [104].

The security problems have pervasive nature, and two kinds of this pervasiveness are recognized. One of them deals with secure coding, the second one is related to crosscutting of the security items. Both can be minimized using the separation-of-concerns principle.

Such pervasive problems as buffer overflow or invalid input data can be avoided by enforcing right coding rules or by using appropriate software development or run-time tools.

The second kind of pervasiveness deals with a common but scattered part of the code that is introduced into the application to meet specific security requirements. For example, the code responsible for events recording in the system logs or access control module can be spread among many classes. In this case the security can be considered as a crosscutting concern since it is related to many classes or functions in the program. This kind of security pervasiveness deals with the structural difference between application and security logic. Other examples of crosscutting concerns are application interactions with the cryptographic library or security manager calls to different places controlled by them.

It should be noted that, by applying the separation-of-concerns principle, more evolvable systems can be built and they can be more easily adapted to unanticipated or changing threats during the product life cycle [104].

Recently, new separation or modularization techniques related to the Aspect-Oriented Programming [4] have been developed. Generally, the separation of concerns entails partitioning the program into distinct elements that overlap in functionality as little as possible and encapsulate these concerns into separate entities with the use of such methods as procedures, packages, and classes. Some specific concerns cutting across many software modules defy this encapsulation, e.g. system logging, errors handling and recovery, and different security concerns. Such concerns should be handled in a specific way, i.e. they are captured in the so-called "aspects". There are two categories of handling the aspects:

- interception-based approaches that intercept certain events in the execution of a program to do some additional processing (implementing the crosscutting concern) before or after the event;
- weaving-based approaches where a weaver tool weaves in the code implementing the crosscutting concern with the other application code, at compile time, load time or even run time.

By separating the security concern as much as possible, two important advantages can be achieved:

- better adaptability of security mechanisms to changes during the life cycle,
- reducing the number of implementation errors, because a well separated concern is easier to capture than a crosscutting concern.

2.1.2. UMLsec concept

Jan Juerjens's works [62-71], summarized in [72], deal mainly with the UML extension, called UMLsec, providing a unified approach to security features description during the secure systems development. The UMLsec can be used to evaluate UML specifications against vulnerabilities. Established formal rules of security engineering can be encapsulated and hence made available to a wider group of developers.

The UMLsec introduces:

- the set of stereotypes (specialized model elements using <<label>>); new meaning to the UML concerning different security features was added, like: <<Internet>>, <<encrypted>>, <<secure dependency>>, <<no-down-flow>>, <<no up-flow>>;
- tagged values i.e. pairs {tag=value} to the stereotyped elements e.g. {confidentiality=TRUE};
- constraints that refine semantics of the stereotyped element, e.g. constraint <<no up-flow>> means that a component prevents up-flow;
- profiles that gather previously mentioned information, e.g. profile <<fair exchange>> of base class package, having tags: start and stop, relating to the constraints: after start, eventually reach stop; this profile is used for enforce fair exchange between a buyer and seller or a sender and receiver, etc. (e-business or network protocols validation);

The UMLsec profiles are used for:

- recurring security requirements offered as stereotypes, like <<secrecy>>, <<integrity>>,

- using associated constraints to evaluate specifications and indicate potential vulnerabilities,
- ensuring that stated security requirements enforce given security policy,
- ensuring that the UML specification provides requirements.

The UMLsec uses specific subsets of the UML:

- use case diagrams, describing typical interactions between a user and a computer system or between different components, used to capture requirements, particularly security requirements,
- activity diagrams for the flow of control (workflow) between system components, explaining use cases in more detail,
- class diagrams to show the static structure of the system, including class attributes, interfaces, operations, signals and relationships,
- interaction diagrams (i.e. collaboration- or sequence-type), presenting interactions between objects by messages exchange (communication protocols explanation, showing directly the handshaking process),
- state-chart diagrams, expressing dynamic behaviour of given objects, allowing "guards", and showing how events may cause states changing or actions triggering,
- package diagrams, used for grouping the system parts into named higher-level units, and specifying objects visibility,
- deployment diagrams presenting underlying environmental aspects of the system, i.e. hardware, software, communication, environment, meeting the system requirements.

The UMLsec allows the patterns that encapsulate the design knowledge in the form of recurring design problems, and consist of the "pattern name", "problem description", "problem solution" and "consequences". The wrapper pattern is a type of pattern used to augment the security of COTS application, working as the intermediate element, ensuring the security of the objects not directly controlled by the developers – instead of calling the original object directly, the wrapper is called and then it passes the call to the original object.

The UMLsec is used to:

- evaluate the UML specification for vulnerabilities,
- encapsulate security engineering patterns, also for designers not specialized in security from early design stages in the system context.

The global formal semantics was provided for the UML subsystems. It incorporates the formal semantics of diagrams included in the subsystems. The introduced semantics allows:

- to express directly internal activities and actions,
- to pass messages between objects or components belonging to different diagrams, by adding a dispatching mechanism for events and handling for actions.

This is the foundation:

- for formal specification of the whole design,
- to solve composition problems in a formal way, to compose subsystems by including others into them,
- to provide tool support, based on this precise semantics, allowing the possibility to simulate specifications or create executable UML specifications.

These works focus on modelling the IT security features and behaviour within the system, and not on the IT security development process. There are no explicit relations with ST or PP development procedures presented in the ISO/IEC TR 15446 standard, and there is no general purpose framework provided for IT security development, according to Common Criteria.

The UMLsec approach is compatible with the presented method of identifying product features and can be very useful for this job.

2.1.3. Engineered Composition (EC) based on the UML

The Shari Galitzer method deals with the composability problem using the CC and the UML. His works are focused on modelling complex security-related products, consisting of other security-related products, evaluated or not, requiring advanced composition methods. The dedicated UML-based specialized framework was provided, not fully compatible with the ISO/IEC TR 15446 standard.

The paper [51] presents an approach, being an extension of Common Criteria, called Engineered Composition (EC), focused on effective composability and verified during the Critical Infrastructure Grants Program from the NIST (National Institute of Standards and Technology). The objective of this work was to achieve the ability to predict that a composition of components will have the desired properties without unpredictable side effects. The composition problem was considered as a modelling problem with security as its subject domain.

It should be noted that according to the CC paradigm the components are PPs and STs and the composition is an evaluated system that reuses the components. The three Common Criteria decomposition approaches were analyzed:

- component TOEs may be combined into composite TOEs in a PP or ST,
- component TOEs may be combined to build a more complex system by users who rely on the TOEs IT environment requirements,
- the TOE may be a system without regard to its component parts.

The EC method seems to be especially convenient for large systems that require distributed development. It provides a means to develop PPs or STs for ensuring that components are combinable and comparable.

The main EC features are:

- top-down decomposition approach,
- based on principles and techniques of information systems object-oriented modelling (UML).

The EC applies these modelling methods and tools to the development of the composed security-related systems according to the Common Criteria, facilitating [51]:

- "the expression of the security requirements of both a system and its components:
  - capturing the security requirements of a component;
  - capturing how the component contributes to the system security requirements;
  - exposing security issues that result from aggregation of the components;
- making combinable and comparable independently developed PPs and STs;
- developing PPs more efficiently by:
  - making PPs 'refinements' for specific environments rather than a 're-creation' for each specific environment;
  - requiring fewer PPs to meet the users' needs;
- developing STs more efficiently by:
  - making STs combinable so it is possible to claim compliance with multiple PPs; currently this approach is assumed as not practical enough;
  - being able to define a scope for an ST that is a component of a set of requirements and being able to reuse those requirements in the ST.
  - reusing PP specifications for an ST that is a component of a PP;
- supporting the fluid nature of systems."

The Engineered Composition method encompasses the following elements:

- standard modelling principles and techniques adopted from system engineering,
- CC paradigm extensions by constructs, like: framework, collaboration, interface and aspects,
- evaluation criteria for the model constructs properties, like: separation of concerns, cohesion and coupling,
- evaluation criteria for rigorous enforcement of the consistent terminology for the CC artefacts and documents.

The EC approach is based on the following concepts: patterns, aspect-oriented modelling methods [4], and interfaces.

Generally, patterns [36] are used in the UML to solve a common problem in a common way. The EC uses two kinds of patterns:

- frameworks – to capture architectural patterns that specify the structure and behaviour of the entire system,
- collaborations – to capture design patterns that specify a set of abstractions that work together to carry out a common and interesting behaviour.

Aspect-oriented modelling can be considered as a newly emerging extension to object-oriented modelling methods to better achieve separation of concerns for complex systems. It is especially convenient to express the complication of cross-cutting concerns.

During the designing process the decomposition took place around concerns, splitting a system into manageable and comprehensible parts. The EC method distinguishes the cross-cutting concerns being the parts that do not decompose neatly. In the composed systems the cross-cutting concerns are properties that apply across several functions or components within a system. They can range from high-level security notions like audit, cascading risk or 'whole-program effects' to low-level notions like redundant copies of data for fault tolerance or synchronization policies, e.g. concerning locking protocol [51]. Also authentication and audit can be good examples of the cross-cutting concerns.

Interfaces are a collection of operations that are used to specify a service of a class or a component, and represent the major seams in the system that permit component parts to change independently [36]. In the EC method "interfaces provide the basis for the recursive property of a system where collaborations and components of a system are themselves made of component parts. The interfaces are represented in the component view of the model. This view exposes the seams of the system and the glue that binds the components. The interfaces that are realized by the cross-cutting properties of the system and represented by the aspect-oriented constructs are called join points" [51].

The EC *framework* comprises the followed parts:

1. A domain framework definition including:

- text description of general technologies and services of the domain,
- security services and capabilities of the domain,
- domain trusted and untrusted users (within models referred to as actors),
- domain-specific terminology.

2. A framework model, capturing 3 levels of decomposition and 2 views of interest.

The EC *framework* encompasses three kinds of inter-connected models expressing a static view of the system from a perspective which supports composition of its component parts.

The models are:

- a system-level object model – a set of *classes* organizing *objects, attributes, operations*, relationships, and semantics of the system and establishing its vocabulary,
- a set of object models for the *collaborations*, i.e., services identified in the domain *framework* definition, inheriting information from the system level, with added refinements and details relevant to the *collaborations*,
- a *component* model that is the representation for the physical seams of the system, inheriting information from the *collaborations,* with added refinements and details relevant to the *components*, created from modelling the *interfaces*.

The two specified *views* of the EC *framework* are:

- the threats and policies *view*, showing the relationship between the threats and policy rules, that implies the requirements, and *objects* and *components*.
- the requirements *view,* taking into account *attributes* of the *objects* and *components*.

3. A new CC composition class of requirement components, capturing and organizing special composition requirements and dealing with:

- adequacy of the composition attributes of the model;
- an interface specification attribute;
- an attribute dealing with the composition risk.

4. A section for administrative information comprising:

- procedures for framework life cycle management;
- profile specifications or references identified by policy requirements;
- policy information on STs claiming compliance with the framework.

The presented above EC *framework* needs to have its formats established. Three formats for data presentation were considered:

- a modified PP format,
- the UML or other modelling method format,
- a hypertext document.

The EC *framework* provides the templates for constituent PPs and STs. Development errors may occur and they can involve the exponential error effect. For this reason, the quality of the EC *framework* must be under control. Therefore an additional APE-type assurance requirement, i.e. additional to those used for the PP evaluation, to evaluate the quality of the framework was introduced. During the evaluation the following two aspects should be considered:

- checking if the EC framework presents a coherent set of security services and capabilities, using suitable CC defined assurance requirements,

- checking if the EC framework and its models are valid from a modelling perspective, by establishing the quality models and its associated metrics, and verifying the basic object modelling principles, like: separation of concerns, collaboration, encapsulation, aggregation and cohesion.

Due to the evolving nature of IT systems the life cycle support for the EC *framework* was adopted from the information systems engineering modelling industry.

A very important issue for the created frameworks, such as the EC *framework,* is the consistency of the terminology. Any occurring discrepancies influence the quality of the framework.

The author has focused on the creation of a supplement to the general CC development framework, especially convenient for solving composition/decomposition problems of the complex IT system with respect to its IT security, rather than on general purposes of the CC development framework creation. The EC method is based on the CC UML *framework*, ensuring that the component PPs and STs for the system are combinable and comparable. It is focused on solving composability problems of large distributed systems.

2.1.4. UML with the B-method

The B-method [34] is a set of mathematically based techniques for the specification, design and implementation of software components. It covers the complete software life cycle, from the specification of the requirements, through the design and its refinement, to implementation, and finally code generation and maintenance. The resulting code can be proven if it is consistent with the original specification. The specification is the abstract representation of the requirements, which express what behaviour is required rather than how to ensure that behaviour. The same rule is applied during IT security product specification.

The formal methods, like B-method, allow to describe behaviour, but within the considered application domain. While implementing this specification, the formal method produces proof obligations that represent the complete set of tests. They confirm that there is no inconsistency between the behaviours of the specification and the design.

The B-method is an object-based method. Systems are modelled as sets of abstract machines which are interdependent and described with the use of the introduced Abstract Machine Notation (AMN). AMN ensures a uniform notation for all levels of description, from specification, through design, to implementation. It encompasses:

- states, comprising a set of variables constrained by an invariant, expressed with the use of notions, like: sets, relations, functions, sequences etc.,

- operations on those states, modelled with the use of pre-conditions or post-conditions; operations may change the state, while maintaining the invariant, and may return a sequence of results.

The general Machine Structure [92] encompasses the following elements:

```
MACHINE name set and numeric parameters
CONSTRAINTS predicate
INCLUDES/SEES/USES machine parameters
SETS names
CONSTANTS names
PROPERTIES predicate
VARIABLES names
INVARIANT predicate
INITIALIZATION substitution
OPERATIONS operations
END
```

The safety or integrity conditions, dealing with the integrity or consistency of the information modelled by the state of a machine, are controlled by the invariant of the machine, for example: “given variable must be a negative number”. The obligation regarding the invariants ensures that the invariant is true before an operation is invoked. It is also the duty of the operation to ensure that the invariant is true after the operation.

The precondition of the operation controls the invariant by capturing, before the operation, all combinations of the state, as well as operation arguments, both of which may break the invariant after the operation.

The B-Method allows:

- to structure large designs,
- the reusability of specification models and software modules,
- consistency of specification checking (preservation of the invariant),
- correctness of designs and implementations checking (correctness of data refinement and correctness of algorithmic refinement).

The B-method is willingly used by the engineers community due to the used formal notation that looks like a simple pseudo programming notation and is rather easy to learn. The existing computer-based tools (B-Toolkit/B-Core, Atelier B/Steria), which are the implementation of the B-method, are used to write, verify and maintain software, including project documenting, formulation of proof obligations, automatic proving, automatic control of the dependencies within complex systems and large libraries of mathematical rules.

Formal methods are used in various mission critical applications, like train or flight control systems and advanced access control systems that use smart cards. AMN is a state-based formal specification language in the same school as VDM and Z (J-R. Abrial).

The work [78] presents an overview of techniques used to get higher EALs, according to the CC standard, for Java smart cards. These techniques are UML- and B-method based, in contrast to [74] which is UML- and EDEN-based (see section 2.1.5).

With the UML approach (semiformal) the following means are used (assurance components belonging to the families responsible for the TOE development, encompassed by the ADV class – *Development assurance requirements*):

- for the SPM (Security Policy Model) – a correspondence matrix of entry parameters (based on use cases) and security policy issues,
- for the FSP (Functional specification) – sequence diagrams to express interactions between actors and systems,
- for HLD (High level design) – collaboration diagrams to express interactions between classes and objects,
- for LLD (Low level design) – sequence diagrams showing modules interrelationships or state diagrams showing modules internal working – depending on the applied granularity,
- and for RCR (Correspondence relation) – a correspondence matrix for abstract representation and refinement.

With the B-method (formal) the following means are used (depending on the existing assurance components within the EAL CC package):

- for the SPM – security automates corresponding to security functions with policies applied as transition constraints,
- for the FSP and HLD – B-model architecture,
- for RCR – B-machine refinement (transformation of the abstract machine into a more concrete one) and refinement proof.

The B-method can be considered as the formal extension for the semiformal UML language used for the higher EAL products, especially for smart cards systems. This approach is focused on the products, not on the IT security development process according to the CC. The B-method can support the framework presented there.

2.1.5. UML method supported by the EDEN formal language

The works concerning the development of an advanced Java smartcard meeting the highest EALs [74] are based on the EDEN specification language, the UML framework and a tool called TL FIT [100]. The EDEN language, having Java-like instructions, is used for logical specification of state changes and events.

The CC requires semiformal approach to EAL5 (designing and testing) and EAL6 (designing, testing, verification), but products gaining EAL7 should be formally designed, verified and tested. For these reasons, special components of the ADV assurance class were defined in the Common Criteria, grouped by families: SPM (*Security Policy Model*), FSP (*Functional specification*), HLD/LLD (*High/Low level design*), IMP (*Implementation*) and RCR (*Correspondence relation*). These representation levels (i.e. families) can be expressed informally, semiformally or formally. The TL FIT framework enables a semiformal description based on the UML, but the EDEN language is the extension of the formal representation. This approach is still under validation through industrial case studies (Java smartcard) and the CC evaluation laboratory works are underway. The below mentioned TL FIT tool (section 2.1.10) is a good example of specialized framework implementation.

This approach can be also considered as the formal (EDEN) extension of the semiformal UML, used for the higher EAL products, especially for smart cards systems. This concept is focused on the products, not on the IT security development process according to the CC, and there is no inconsistency between the concept and the methodology presented in the monograph.

2.1.6. UML method supported by the OCL formal language

The OCL (*Object Constraint Language*) [103] is a formal textual assertion language used for precise modelling with the UML, facilitating design by contract. It is free of side-effects. The basic constraints, represented in the UML as stereotypes, are: *invariants*, *preconditions* and *post-conditions*. The OCL is a kind of trade-off between strict formal mathematical notation, can be used by a limited group of developers only, and is an easily used natural language. The OCL has a well defined mathematical basis, but is friendly to systems engineers and programmers.

The paper [75] focuses on expressing Common Criteria security requirements as constraints using the OCL in a UML domain model to improve the embedded software development process. It was noted that this software is usually implemented in low-level languages, without formal requirements and models, especially without security requirements.

Implementation requirements were provided by using model-based architecture developed by OMG (*Object Management Group*) [85] which considers multiple levels of abstraction:

- computation independent domain model,
- platform independent computational model,
- platform specific implementation model.

The first one, a domain model, expresses domain knowledge, e.g. concerning smart cards. It also implements business requirements of a domain, e.g. how to use these smart cards. The Common Criteria functional and assurance requirements (described by CC components) are used to support domain modelling and are expressed with the use of the OCL language. The OCL is used to specify requirements as the constraints for domain model attributes and operations, for example:

"*FCS_CKM.1: Cryptographic key depends on algorithm and key size*" can be presented using the OCL as:

```
--Key constraints
self.key=KeyGeneration(a: algorithm, s: keySize)
```

The security requirements which are not directly related to domain attributes or operations, e.g. those dealing with audit data storage and management, belong to the computational model. The OCL language will be broadly used in the framework introduced there to extend the UML approach and achieve higher modelling preciseness.

2.1.7. Using AutoFOCUS within the security domain

AutoFOCUS [3] is a model-based tool for the development of reliable embedded systems. It supports different steps of the top-down development process, providing many views of a component-oriented model:

- structural view in the form of a system structure diagram, considering: hierarchy of components, ports and channels for message exchange,
- interaction view as sequence diagrams, extended event traces, message sequence charts,
- behavioural view, as a state transition diagram defining behaviour of components by means of transitions with their pre- and post-conditions, and patterns of exchanged messages,
- data view, presenting definitions of values and functions, encompassing user-defined data.

AutoFOCUS was used in the implementation of many responsible technical systems, including smart cards systems. AutoFOCUS can be extended with information security features, allowing seamless consideration of security aspects in the development process that can be supported in modelling, simulation, consistence checking, code generation, validation, verification, and testing.

2.1.8. SPARK – a programming language for high integrity systems

The paper [47] considers a description language called SPARK, based on ADA, widely used in the aerospace and rail industries [98], suited to the development of high-integrity systems. The paper shows that SPARK can be also suitable for ITSEC [61] and the Common Criteria secure systems development process.

The author notes that the use of formal methods in the specification of systems is widely spread, while the use of formal implementation languages is not well known. The implementation stage can introduce ambiguity that can be a source of serious security breaches, like "buffer overflow". The widely spread programming languages have many ambiguity sources, for instance: float point number operation, libraries, dependencies on the compiler and target processor, evaluation order of expressions, order of associations of parameters, subprogram parameter passing, etc.

Growing potential and internal complexity of recently used programming languages are not convenient for designing high-integrity systems in which all details must be under control. For this reason, SPADE ADA Kernel, called SPARK, and tools associated with the language, were introduced.

While developing SPARK, some elements of the ADA environment were removed to assure no ambiguities and simplicity, but the language was left rich enough to describe real systems. SPARK can determine statically whether a program conforms to the language rules. SPARK programs are verifiable. They have built-in mechanisms eliminating the possibility of run-time errors caused by exhausting finite resources such as time and space. SPARK is designed to be compiled with no supporting run-time library. Many ADA features were removed, like *goto* statement, aliasing, recursion, user-defined exceptions, or simplified – mostly those dealing with "types".

The following features of SPARK are convenient for the development of secure systems [47]:

- program-wide, complete data- and information-flow analysis;
- proof of correctness of SPARK programs is achievable; this allows to extend the formality in the design and specification of a system through its implementation;
- proof of the absence of predefined exceptions (for such things as buffer overflows, dividing by zero); it offers strong static protection against a large class of common security flaws;
- SPARK can be compiled without a supporting run-time library.

The paper presents experiences with SPARK used for designing specific Certification Authority (CA) for the smart cards Multi-Application Operating System (MULTOS) environment, to meet the standard ITSEC E6 level (corresponding CC EAL7).

MULTOS CA was developed with the use of different tools, however it was SPARK that was applied (30% of rough code) for the "security kernel" of the tamper-proof software. ADA95 (30%) was used for infrastructure (concurrency, inter-task and inter-process communication) bindings to ODBC and Win32. C++ coding (30%) was used for GUI components, C (5%) for device drivers and SHA-1 implementation, while SQL (5%) for database stored procedures.

2.1.9. Emerging Common Criteria implementations

SECOQC (*Development of a Global Network for Secure Communication based on Quantum Cryptography*) [96] is one of the UE 6th Framework Program projects in the trust and security area. Its aim is "to specify, design, and validate the feasibility of an open, Quantum Key Distribution (QKD) infrastructure dedicated to secure communication as well as to fully develop the basic enabling technology". One of its subtasks is that the CC ought to create the basis for the IT security certification of QKD applications.

The CC is focused on providing „the systematic proof of the existence and effectiveness of the intended security properties, from quantum physics through key establishment to the distribution of keys in a network". It is assumed that the assurance of the QKD applications will be gained in the same way as the assurance for other IT applications, i.e. by means of specification and evaluation by independent testing laboratories. This project shows that the mature and general-purpose Common Criteria methodology goes towards the emerging IT technologies, i.e. cryptography based on quantum physics, though no papers presenting results of this subtask were found up until now.

2.1.10. Computer-aided tools

There are three main groups of tools designed for IT security developers and evaluators. The first one supports the Common Criteria IT security development process in a less or more detailed way [43], [90], [101]. These applications help to manage design stages and related documentation. All tools have CC functional and assurance components implemented and allow to define mnemonic descriptors expressing IT security features, called "generics". Some tools [43], [90] have only a basic set of generics predefined with relations between one another, some offer the possibility of using only the generics defined by the user [101]. These tools are designed rather for lower EALs (informal specification) and can be useful for commonly used products, like COTS. They encompass both developers' and evaluators' tools. The basic features of the tools belonging to the first group are:

- structured production and edition of Security Targets or Protection Profiles,
- generation of Common Criteria documentation,
- consistency verifications and traceability.

The second group of tools, designed for higher EALs, is enhanced but also application-specific (usually for Java smartcards). The tools focus mainly on proper implementation of the ADV class, based on semiformal or formal approach, like the UML, OCL, B-method and tools [100] being an extended version of the TL SET [101]. TL FIT is a unique environment shared by designers, developers and evaluators, providing them with a variety of modules for textual, graphical, semiformal and formal specification, tools for verification and document generation – integrated within the above mentioned TL SET suite of tools. TL FIT also supports the evaluation documents management (UML-based). It provides automatic generation of the ADV class assurance components and construction of security policy definition.

The third group, designed for the evaluators, supporting the implemented evaluation scheme, like [56], will not be considered there. These features are offered by [90] and also by ITSDF-tool, which is the implementation of the methodology presented there.

It should be noted that some tools are developed as a part of know-how of the IT development or evaluations laboratories, and for this reason their description as well as the tools are not often publicly available.

2.1.11. Security engineering environment – around the performed overview

From the general point of view the security engineering ought to deliver IT products or systems with the assurance at the expected level. These deliverables are used to build IT systems which provide information processing and different forms of IT services for organizations. Using such deliverables evaluated against this level, i.e. certified, allows to achieve the information and services security more easily, though does not guarantee this. The information security always needs the right management. For this reason, specific methodologies were developed, mainly around the ISO/IEC 27001 standard [58], presented also in a few works by the author: [8-9], [11], [17-18], [23], [25], [31].

Today's global organizations often belong to critical infrastructures for which assurance is extremely important and quite new R&D challenges exist, e.g. concerning global dimension, new forms of threats, inter- and intra-dependencies, enabling cascading effects [48], [16], [20-21]. The Common Criteria methodology seems to be promising in these applications.

The draft report [84] presents results of the work on OSMOSE (*Open Source Middleware for Open Systems in Europe*), a security framework that uses the Common Criteria to achieve the assurance. The basic framework entities are defined for: security policy, roles, privileges, credential, security service, permissions, resources, and certification authority. Different open platforms are mapped. The number of certificates of open source products is still growing. Additionally, please note the above mentioned recent research and development concerning the quantum cryptography which take into consideration the Common Criteria approach [96].

The above overview also shows the growing importance of both the Common Criteria and the UML methodologies. The UML has recently become a very important tool in the realm of the information security and systems safety domains. Let us present a few other examples.

The paper [55] uses the UML to present the behaviour of the system of systems belonging to high-integrity real time systems (military, aerospace applications). Particularly, it shows how the safety policy is decomposed into rules that an individual system ought to satisfy. The paper [52] discusses the UML-based trust ontology. The discussed trust cases are intended to justify and support claims concerning trustworthiness of IT applications and services. The report [35] presents a CORAS-based approach to the risk modelling. CORAS is a graphical language and tool. Graphical elements are UML stereotypes representing different security issues, like assets, threats, risks and safeguards, playing the key role during the risk modelling. This report demonstrates the use of CORAS for modelling threats in relation to Microsoft® technologies, like Web Services, ASP.NET, SQL Server, Active Directory and Smart Cards, though CORAS is suitable and broadly used in many other applications.

Ontological approach to the CC methodology is presented in the papers [50], [105], and in the recent paper of the monograph author [106].

2.1.12. Technology overview summary

The conclusions of the above review show that there are three main groups of researches based on the UML, focused on:

- solving complex composition problems, like the Engineered Composition (EC) method [51],
- modelling behavioural aspects of IT security-related products using a UML subset called UMLsec [72],
- solving application-specific problems, like Java smartcards, for higher EALs [74].

All of them provide different types of supporting tools. These tools need better integration with the UML world as well as improving their basic functionality offered for developers.

The above surveyed frameworks and tools are focused on one side of the dedicated issues only: either on the framework representation or on the developed security product modelling with the use of the UML. There are no IT security development frameworks and tools allowing to harmonize both.

Some works postulate: creating unified assurance frameworks mostly for COTS, implementing risk management features and evaluating non-IT components [81], [77] issuing products of low-cost evaluation [82], improving compliance with information security management standards [86], [73], [37], [12] and a broad use of the XML language.

To sum up, there are not any general-purpose, CC development process-centric, modelling methods for common security-related products based on the UML and CC, and compliant with the ISO/IEC TR 15446 approach.

2.2. Developers’ needs with respect to the IT security development support

The development of the IT security-related products or systems is specific and more formalized [38]. There are two issues to be distinguished at the beginning:

- the IT product or system (i.e. TOE) development process, which has broader meaning than the IT security development, and encompasses all developer's efforts from identification of users' requirements, product/system development including IT security development, to final deployment,
- the IT security development process related to the TOE, i.e. ST/PP elaboration, which is a middle and very important part of the above efforts – the ST/PP elaboration starts from the identification of IT security needs and concerns (using the preliminary IT product/system specifications) and completes at the requirements for the security functions elaboration (implemented later in the product/system).

The IT product or system development process gives an input to the ST specification elaboration, and later, when the ST is ready, uses the ST as the basis for the further TOE development.

The IT security development defined as part of the CC methodology is common but IT development methods may be different and specific for hardware, software, and network equipment designs. In the software engineering the UML, OCL, and Java approaches are growing. Hardware description languages (e.g. VHDL, Verilog) and related aiding tools are used for hardware products. The IT development process may be supported by the methods specified in a framework for IT security assurance [59]. It is important to note that no restrictions on the TOE development were specified.

Usually, the TOE development encompasses two kinds of efforts, compliant with the elaborated IT security specification:

- top-down efforts – refinement of the design specifications: from security requirements, through functional specification, high-level design specification, implementation, to low-level specification; the higher the EAL, the more detailed specification required;
- bottom-up efforts – correspondence analysis and integration testing in reverse direction; and, on this basis, modification of the specification on a more general level.

This monograph concerns better formalization of the IT security development process however it does not concern IT (hardware, software, systems) development and IT security evaluation processes, though these three complex issues are strongly related with one another and mutually supportive. The IT security development of products or systems according to the CC methodology was shown in the Fig. 2.1 and Fig. 2.2. This process consists of four (three for PP) main steps:

- establishing security environment, defined by sets of assumptions, threats and organizational security policies (OSP), worked out during an analysis: TOE assets, purpose and physical environment,
- setting security objectives – for the TOE and its environment, using CC components catalogues and analyzing the above objectives, working out the sets of functional and

assurance requirements for the TOE and for the environment, using functional and assurance requirements, preparing the TOE summary specification (TSS) – for the ST specification only.

Different ways of creating the ST or PP main specifications were shown. The TOE can be designed:

- straight on the basis of consumer needs,
- using consumer needs, and additionally in compliance with given PPs,
- based only on the requirements defined within the previously evaluated PPs.

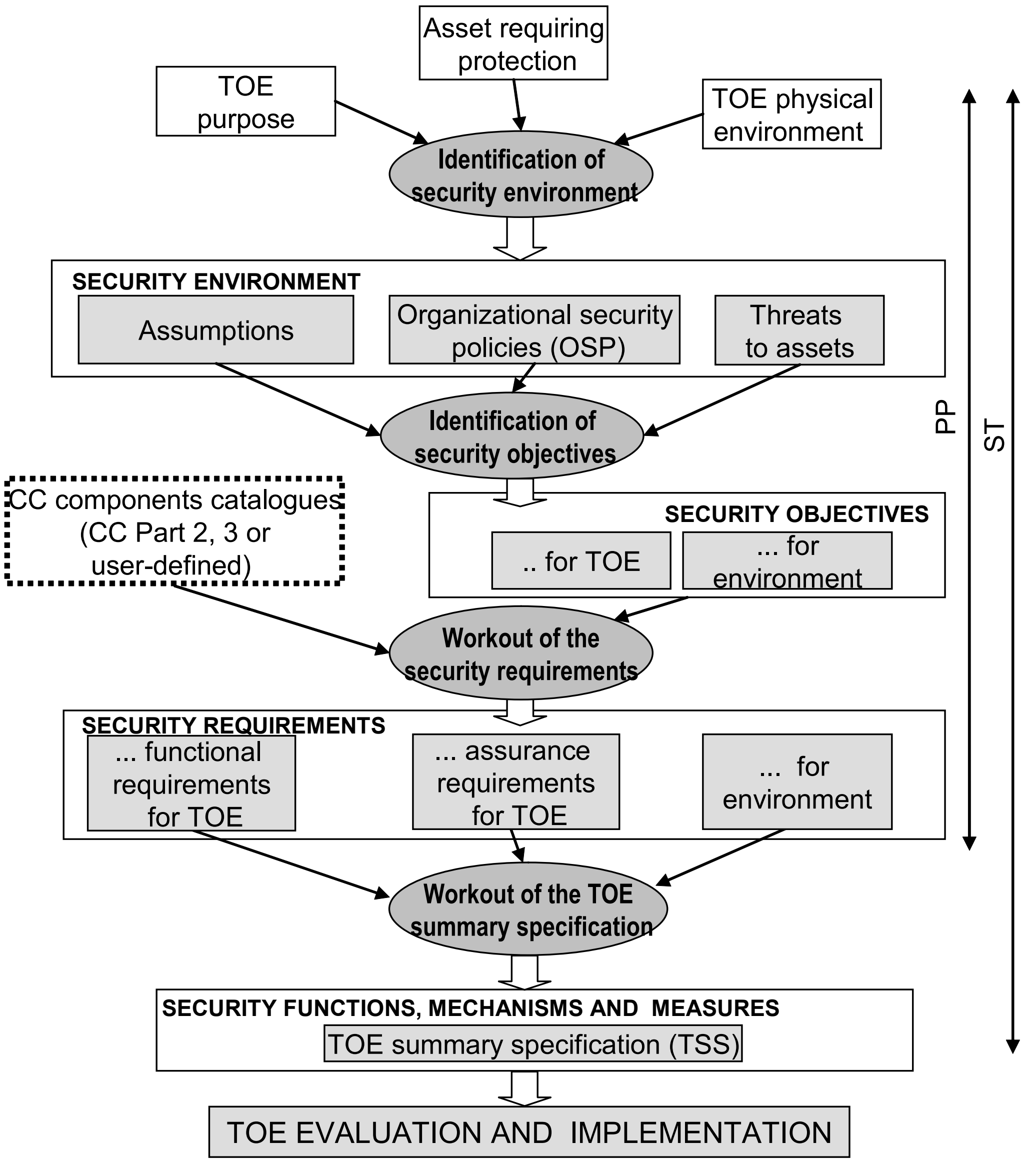

Asset requiring protection
TOE purpose
TOE physical environment
Identification of security environment
SECURITY ENVIRONMENT
Assumptions
Organizational security policies (OSP)
Threats to assets
Identification of security objectives
CC components catalogues (CC Part 2, 3 or user-defined)
SECURITY OBJECTIVES
.. for TOE
... for environment
Workout of the security requirements
SECURITY REQUIREMENTS
... functional requirements for TOE
... assurance requirements for TOE
... for environment
Workout of the TOE summary specification
SECURITY FUNCTIONS, MECHANISMS AND MEASURES
TOE summary specification (TSS)
TOE EVALUATION AND IMPLEMENTATION
PP
ST

Fig. 2.1. General scheme of IT security development process
Rys. 2.1. Schemat ogólny procesu konstruowania zabezpieczeń

On this basis a more detailed scheme of elaborating Protection Profiles and Security Targets was worked out, presented in the Fig. 2.2. After each stage a rationale process is required and the compiled security model is getting more and more precise. Different means and methods are used for security model expression at these stages. Please note that the Common Criteria include semiformal specification means, called functional and assurance components, though they can be used only on the requirements specification level (Table 2.1).

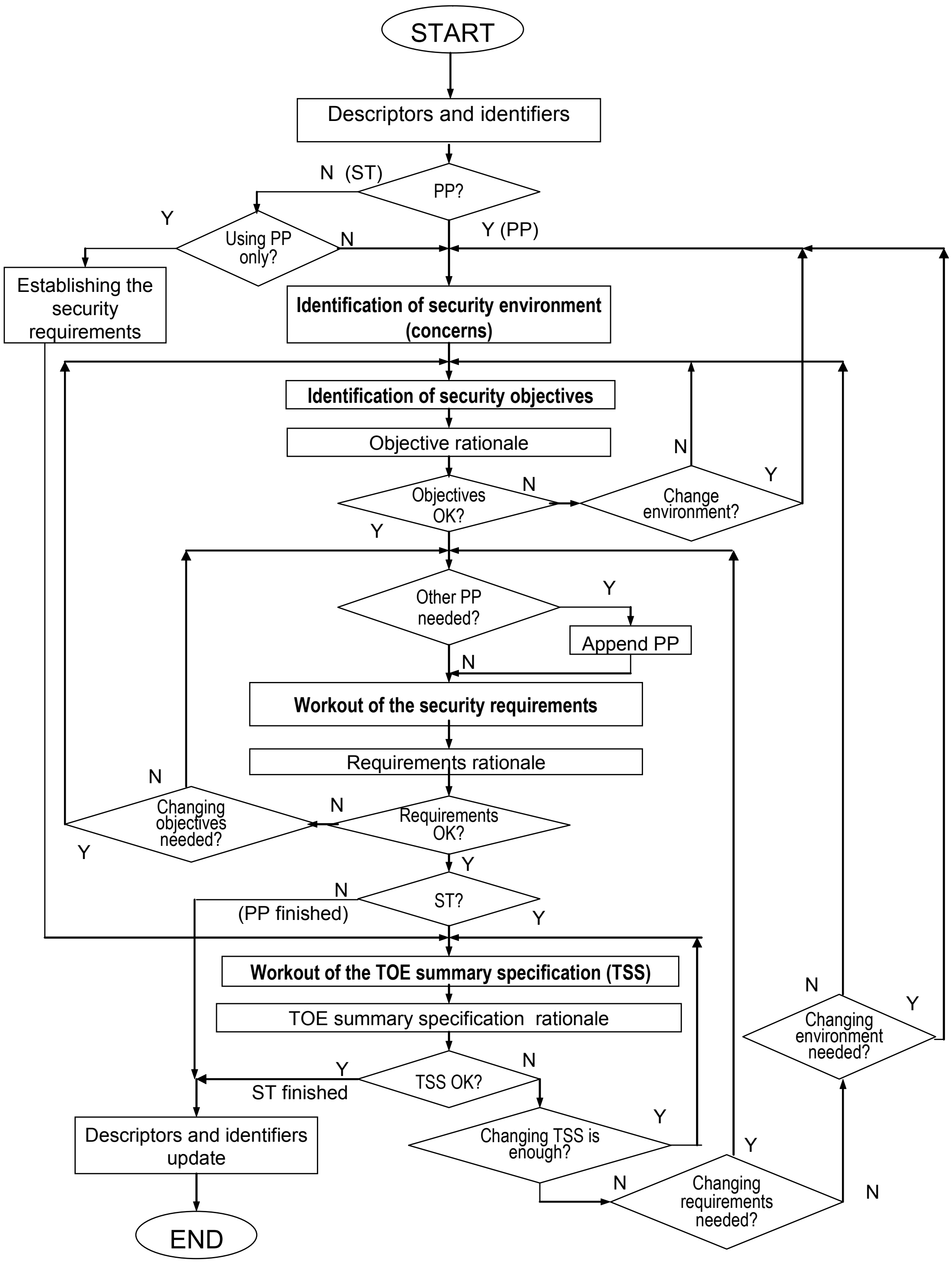

START
Descriptors and identifiers
PP?
N (ST)
Y (PP)
Using PP only?
Y
N
Establishing the security requirements
Identification of security environment (concerns)
Identification of security objectives
Objective rationale
Objectives OK?
N
Y
Change environment?
N
Y
Other PP needed?
Y
Append PP
N
Workout of the security requirements
Requirements rationale
Requirements OK?
N
Y
Changing objectives needed?
N
Y
ST?
N
(PP finished)
Y
Workout of the TOE summary specification (TSS)
TOE summary specification rationale
TSS OK?
Y
ST finished
N
Changing TSS is enough?
Y
N
Changing requirements needed?
Y
N
Changing environment needed?
N
Y
Descriptors and identifiers update
END

Fig. 2.2. Security Target (ST) and Protection Profile (PP) development process
Rys. 2.2. Konstruowanie zadania zabezpieczeń (ST) i profilu zabezpieczeń (PP)

Functional requirements, designed for safeguards functionality modelling, are expressed with the use of functional components taken from [39], according to the functional paradigm defined there. Assurance requirements, designed for assurance (safeguards "reliability and credibility") modelling, are expressed with the use of assurance components taken from [40], meeting the declared EAL and evidences. The CC standard provides hierarchically ordered means (class –> family –> component) for specifying functional and assurance requirements only. For the other model stages precise verbal descriptors are used, sometimes formed as simple generics.

Table 2.1

The specification means at any stage of the IT security development process

IT security development stage	Used specification means	Proposed specification means
TOE description	Informal (textual)	Informal (textual), semiformal UML models
Security environment (problem)	Informal (textual, simple generics)	Semiformal, enhanced generics
Security objectives	Informal (textual, simple generics)	Semiformal, enhanced generics
Requirements:		
• functional for the TOE	Semiformal CC functional components	Semiformal CC functional components
• assurance for the TOE	Semiformal CC assurance components	Semiformal CC assurance components
• functional for the environment	Informal (textual, simple generics)	Semiformal, enhanced generics or semiformal CC functional components
• assurance for the environment	Informal (textual, simple generics)	Semiformal, enhanced generics or Semiformal CC assurance components
Trusted security functions	Informal (textual, simple generics)	Semiformal, enhanced generics

Generics, having features comparable with the semiformal CC components, called there enhanced generics, were introduced for the first time in [13], but this monograph will provide a more comprehensive and well formed definition. The enhanced generics have the following features:

- possibility of parameterization,
- operations on generics – iteration, refinement, assigning value to a parameter or leaving it uncompleted,
- defining any generic on the basis of the other (derivation), grouping generics by their domains of application,

- assigning attributes, performing operations,
- building generics chains – proposing solutions to elementary security problems.

Providing the developers with the uniform specification language at any development stage allows to implement generics and components as the design library for the computer-aided tool, which makes the development process easier and more effective. Better efficiency of the IT security development process means the improvement of the specification preciseness and reduction of the development time and cost.

For IT security development, especially computer-aided, two key issues are important for developers:

- defining a general scheme of actions provided, with respect to different options – creating a kind of a roadmap for developers,
- providing developers with means and tools serving for the specification of different aspects of security models created and refined during the development process.

The solution to these issues is the subject of this monograph. While analysing the Common Criteria philosophy and the related development process, the most difficult and complex elements of this methodology were identified. Developers need right support in these matters. For that reason, the following general assumptions were specified:

- design procedure is compatible with the ISO/IEC TR 15446 standard, but significantly extended and expressed more precisely with the use of the semiformal, or sometimes even formal, approach;
- the framework has open general-purpose character, especially convenient for COTS;
- the framework will be provided with the advanced library of the unified specification means, including enhanced generics, having features mentioned above;
- better focus on problem solving due to the possibility of direct design specification with the use of the UML;
- the framework facilitates the selection of the right security items (solutions) to cover other security items (problems);
- there is a risk analyzer built in;
- better support of the design trade-offs dealing with a developed security-related product,
- compatibility with information security management standards, which define the environment for evaluated security-related products in organizations;
- advanced approach: 2-step rationale process, SOF-claims elaboration, managing the assurance requirements of different sources;
- better management of the design documentation, including evidence,
- offering features and facilities for self-evaluation of the project;
- providing better composition support and design reusability.

Better support offered to developers is very important to issue more precise designs in a shorter time.

2.3. General model of the IT security development framework

The IT security development process can be considered in two aspects:

- as a definitional problem – dealing with a statement of the nature of a thing,
- as a methodological one – concerning the way of doing something,

but each of them within the following domains:

- IT systems modelling domain,
- IT security modelling domain.

It has a big influence on the kinds of models that can be created. The UML-based IT Security Development Framework should be open to any kind of IT security-related products or systems that have or do not have their UML model (called there `EUM_EntryUML model`). For this reason, at the beginning of the development process it was necessary to create simple, auxiliary models of these products and systems to capture any relevant product features in the unified manner. To express and solve the definitional and methodological problems within the IT systems modelling domain, it is recommended to use the standard approach, based on the latest UML achievements.

The main objective of this work is to establish a workable structure of the IT security development system, considered as the framework, evolving into successful implementation. The IT Security Development Framework and IT security domain models should comply with the Common Criteria and related standards. To simplify the problem, IT security development can be considered as the creation of the STs or PPs, understood as the superior IT security domain models. To achieve this, both definitional and methodological aspects can be considered in relation to the matter of fact, i.e. designed security-related products (TOE). The definitional aspect encompasses data structures and objects related to the CC terminology, like: threats, assumptions, policies, objectives, requirements, security functions, etc. The methodological aspect deals with coordinated actions attempted at passing from one development stage to other, like: preparing TOE description, IT environment elaboration (called "security problem definition"in the latest CC version), risk analyzing, performing the functional security requirements rationale, prediction of the strength of security functions or their audit needs. The definitional aspect support is provided by CC-defined components [39-40] and user-defined generics. The methodological aspect is described on a very general level in [38] and on a detailed level in [60], but all of them are expressed verbally or semiformally (CC components), without using UML specification means and ways. The use of UML facilities for elementary development actions e.g. decomposition (top-down) and composition (bottom-up), will make the developers' work easier and more effective. They can focus on a limited number of details and how these details influence the whole at the considered level of abstraction, i.e. at the considered development hierarchy level.

By using the UML approach in the security modelling domain, all UML benefits [36] can be achieved. Still, the three major ones are the following:

- easier visualization of STs or PPs structures, behaviours, and relations – at different levels of abstraction,
- elaborating a set of templates convenient to use in any circumstances,
- documenting the whole process and related decisions.

The IT development both according to the CC and UML modelling is well known. However, the question how to merge these approaches effectively is the subject of the recent researches and the challenge for many of them.

The `ITSDF_ITSecurityDevelopmentFramework` class represents the IT Security Development Framework as a whole. It is based on Common Criteria and encompasses three concurrently developed UML-based models (Fig. 2.3):

- the model of the security-related product, called the PM model, presenting its elements, functionality offered for users, product-concerning requirements and working environment – related to the technical documentation.
- the security model of the security-related product, called the SM model, created on this basis of and using the Common Criteria method of specification – related to the security documentation, i.e. the ST or PP, required for evaluators and other CC consumers,
- the security self-evaluation model, called the SEM model, responsible for the developed product or system security features assessment, according to the evaluation scheme and methodology.

The IT Security Development Framework is provided with the security library (`SL_SecurityLibrary` class), which makes the whole designing easier.

The `ST_Elaboration` class is responsible for the ST development process. The `ST_Elaboration` class behaviour will be presented later using the finite state machine (see Fig. 2.6). The `PP_Elaboration` class will not be discussed there as it is similar to the above mentioned `ST_Elaboration`. The classes representing evaluation processes (`STSelfEvaluation`, `PPSelfEvaluation`) will be presented briefly in the chapter dealing with the evaluation.

The model of the security-related product (`PM_ProductModel` class) depends strongly on the domain of application (kind of product). Building such a model corresponds with typical activities carried out during the UML modelling of IT products or systems, but some product or system features have to be emphasized, i.e. those which provide the input for the security model. It could be assumed that for very many commonly used IT products or systems their UML-based models exist and their detail levels are enough.

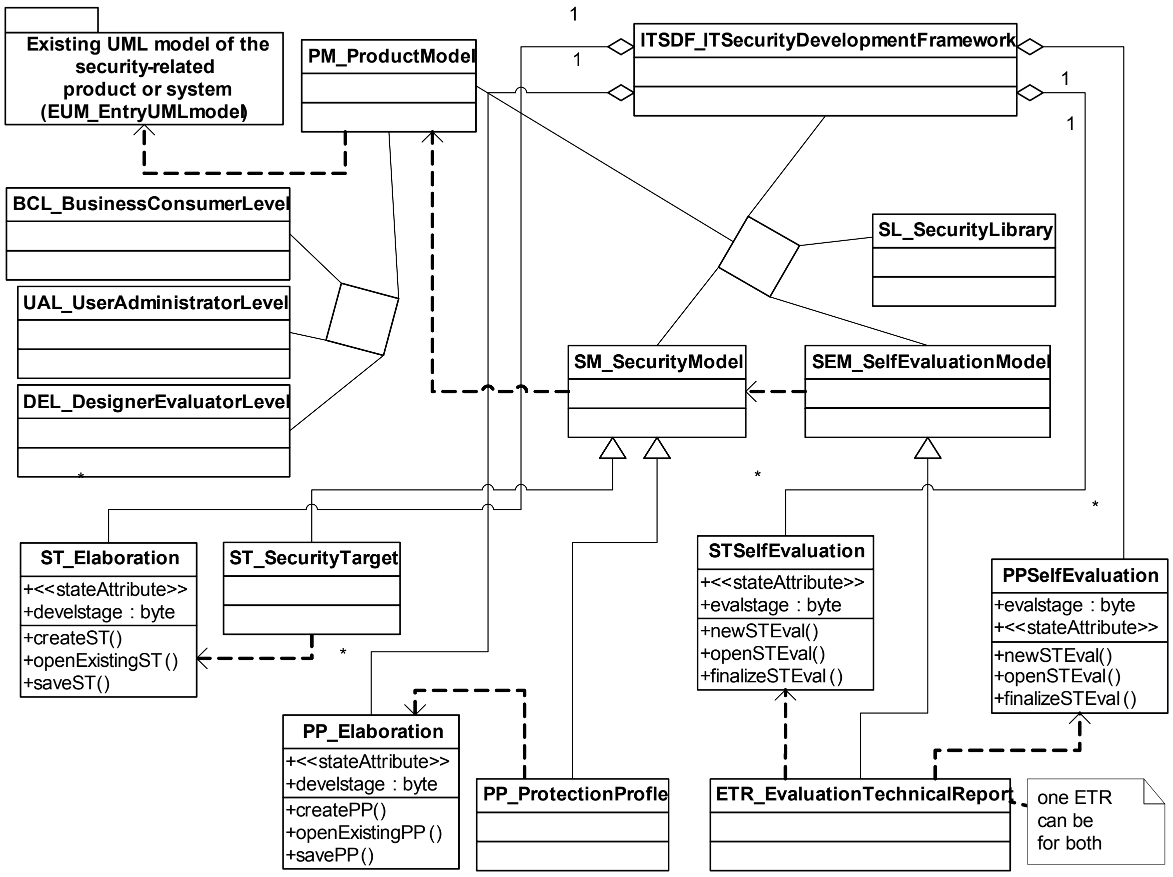


Fig. 2.3. PM, SM and SEM models – general concept of the UML IT Security Development Framework, according to the Common Criteria standard
Rys. 2.3. Modele PM, SM, SEM – ogólna koncepcja szkieletowego systemu konstruowania zabezpieczeń informatycznych, opartego na UML i zgodnego ze standardem Wspólne krytreria

The problem of creating the model of a security-related product is how to transform a freely existing UML model (i.e. `EUM_EntryUMLmodel`) of the product into an auxiliary PM model. When no UML model exists, creating a simple PM model is recommended. It can be developed concurrently with the SM model refinement needs. It was assumed that the PM model, being input to the SM model, should contain the following three basic levels with elements that partially correspond to elements of ST or PP structures, and partially are their simple extensions, presented in a more detailed way in security-related products:

- BCL (`BCL_BusinessConsumerLevel` class) – business and consumer level for sponsors and managers (buyers) – expresses general information about the product or system sufficient to select them on the market to meet specific consumer needs,
- UAL (`UAL_UserAdministratorLevel` class) – user and administrator level – encompasses information for the product or system users and administrators,

- DEL (`DEL_DesignerEvaluatorLevel` class) – designer and evaluator level – represents data exchanged between them during the evaluation process performed in the security labs.

Developers should be able to self-evaluate on the fly the results of their work, making necessary corrections and improvements as soon as possible. For this reason the development process has many feedback loops at different stages and these loops are based on different evaluations of the current results. The most important of these checks can be performed at the final stage of the work almost in the same way as it was done with independent security lab during the official evaluation/certification process.

This possibility is provided by the third part of the IT Security Development Framework presented there, i.e. by the IT security self-evaluation framework built in the ITSDF framework. It is represented by the SEM self-evaluation model (`SEM_SelfEvaluationModel` class). Simplifying the problem, the IT security evaluation process is related to the creation of the formalized, ETR-type document (Evaluation Technical Report) that summarizes its efforts and achieved results. The SEM model allows to imagine how the developer's work will be evaluated by an independent body. Please note that SEM represents the data concerning supportive, internal evaluation made by the developer, the DEL model contains almost the same data, however exchanged between the developer and evaluator during the real evaluation process.

The SM model is compliant with the Security Target (ST) – Fig. 2.4 or Protection Profile (PP) – Fig. 2.5 structures, well defined within the CC standard. The Fig. 2.4 shows classes representing the developed ST data structures and classes responsible for their development. The monograph is focused on the ST elaboration which is a more comprehensive task than the PP workout.

The development of IT security encompasses the following main tasks:

- building the PM model,
- development of the SM model.

It is assumed that the development of PM does not have to be completed, but must be refined enough to provide some input data for the development process of the SM model. The development of SM corresponds to the main stages of creating PP/ST specification and each of the stages will be successively expressed.

The IT development framework has two views corresponding to the above mentioned models:

- horizontal view expressed by the PM model,
- vertical view expressed by the SM model.

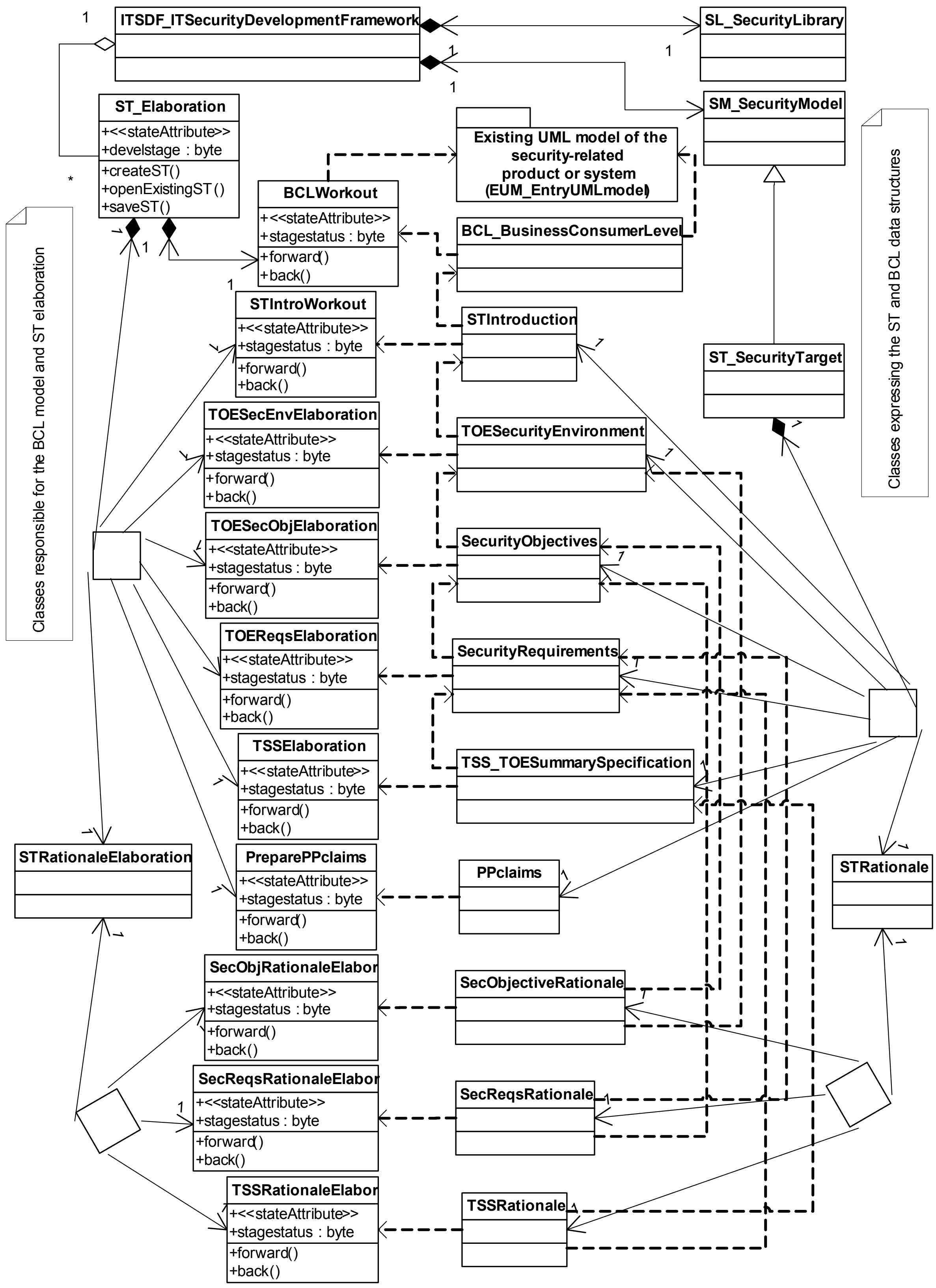


Fig. 2.4. Security Target (ST) and its elaboration
Rys. 2.4. Zadanie zabezpieczeń (ST) i jego konstruowanie

It is recommended to use security engineering principles and methods together with the presented IT Security Development Framework, although none of them is especially preferred.

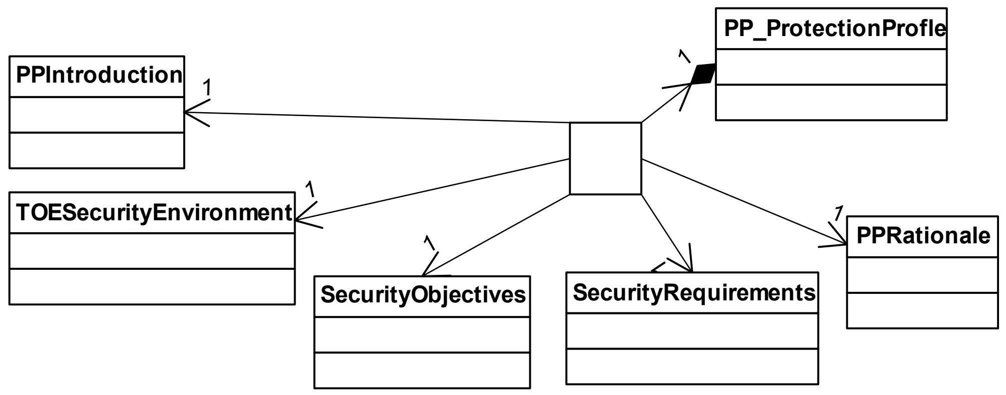


Fig. 2.5. Protection Profile (PP) as the set of classes (aggregation)
Rys. 2.5. Profil zabezpieczeń (PP) jako zbiór klas (agregacja)

The presented ITSDF framework has built-in facilities for capturing interactions and relations between the IT system and its environment, allowing to define the developed products more precisely, giving no restriction on the used development method (e.g. software engineering).

The auxiliary PM model does not have to be a full UML product description. Its precision should be adequate to its possibility to sample all features needed to build an SM model. It can be created on the basis of technical documentation or on the basis of parts of the existing UML products or system models.

2.4. IT security development process as a state machine

The ST or PP are the basic artefacts issued on the output of the development process. This process can be expressed as a simple state machine in which all development stages are states that have their own substates. It will be shown for the ST, as a more complex example than PP (Fig. 2.6). The discussed state machine is the implementation of classes responsible for the ST development, placed on the left part of the Fig. 2.4.

Due to the complexity of the problem, all stages will be refined in the next chapters on separate diagrams (UML notes show their figure numbers), representing the above mentioned internal state machines of particular stages. A simple transition control mechanism, one of many possible ones, was assumed. It is very intuitive because it is similar to the IT security developers' activities. As it is not shown in the Fig. 2.6[2], a short comment will be necessary.

[2] The transition control between the development stages is explained in the Fig. 4.3, as in the first discussed development stage example.

Two enumeration variables are introduced to control the IT security development process. The enumeration type indicating the current development stage is defined for the ST[3] as follows (see the state attribute *develstage* in the Fig. 2.3):

Definition 2.1: Semantics of the state attribute expressing the current IT security development stage – see details about semantics of the enumeration types included in the Appendix B.

The semantics of enumeration type $t_{develstage} \in T_E$ of the *develstage* state attribute is the function:

$I(t_{develstage}) = literals(t_{develstage}) \cup \{\perp\}$.

The following interpretation of literals of type $t_{develstage}$ is assumed:

$I(e_1) = I(BCL)$: "Business/consumer level model elaboration",
$I(e_2) = I(ST\_INTRO)$: "ST introduction elaboration",
$I(e_3) = I(SEC\_ENV)$: "Security environment/concerns elaboration",
$I(e_4) = I(SEC\_OBJ)$: "Security objectives elaboration",
$I(e_5) = I(SEC\_REQ)$: "Security requirements elaboration",
$I(e_6) = I(TSS)$: "TSS elaboration",
$I(e_7) = I(PP\_C)$: "Preparing PP claims",
$I(e_8) = I(SEC\_OBJ\_RAT)$: "Security objectives rationale",
$I(e_9) = I(SEC\_REQ\_RAT)$: "Security requirements rationale",
$I(e_{10}) = I(TSS\_RAT)$: "TSS rationale".

□

The second enumeration type expresses the development status of the given stage, showing overall progress of the whole IT security development process. It concerns any ST (see Fig. 2.4) or PP (Fig. 2.5 – not discussed) stage.

Definition 2.2: Semantics of the state attribute expressing the development status of the given stage.

The semantics of enumeration type $t_{stagestatus} \in T_E$ of the *stagestatus* state attribute is the function:

$I(t_{stagestatus}) = literals(t_{stagestatus}) \cup \{\perp\}$.

The following interpretation of literals of type $t_{stagestatus}$ is assumed:

$I(e_1) = I(ELABORATED)$: "Under development",
$I(e_2) = I(CHECKED)$: "After positive checkings dealing with the specification completeness, coherency, mapping of items, justifications, correctness, etc., performed at the end of the stage",
$I(e_3) = I(CLOSED)$: "All stages are checked positively, rationale finished".

□

Every state (expressed by the *develstage*) has entry and exit actions. At the exit of the state, a general review of the specification issued during the current stage is performed

[3] For the PP, not discussed there, the *develstage* can be defined similarly, expressing each PP development stage.

(checking: completeness, coherency, items covered by other items, issued justifications, correctness, etc. – discussed later).

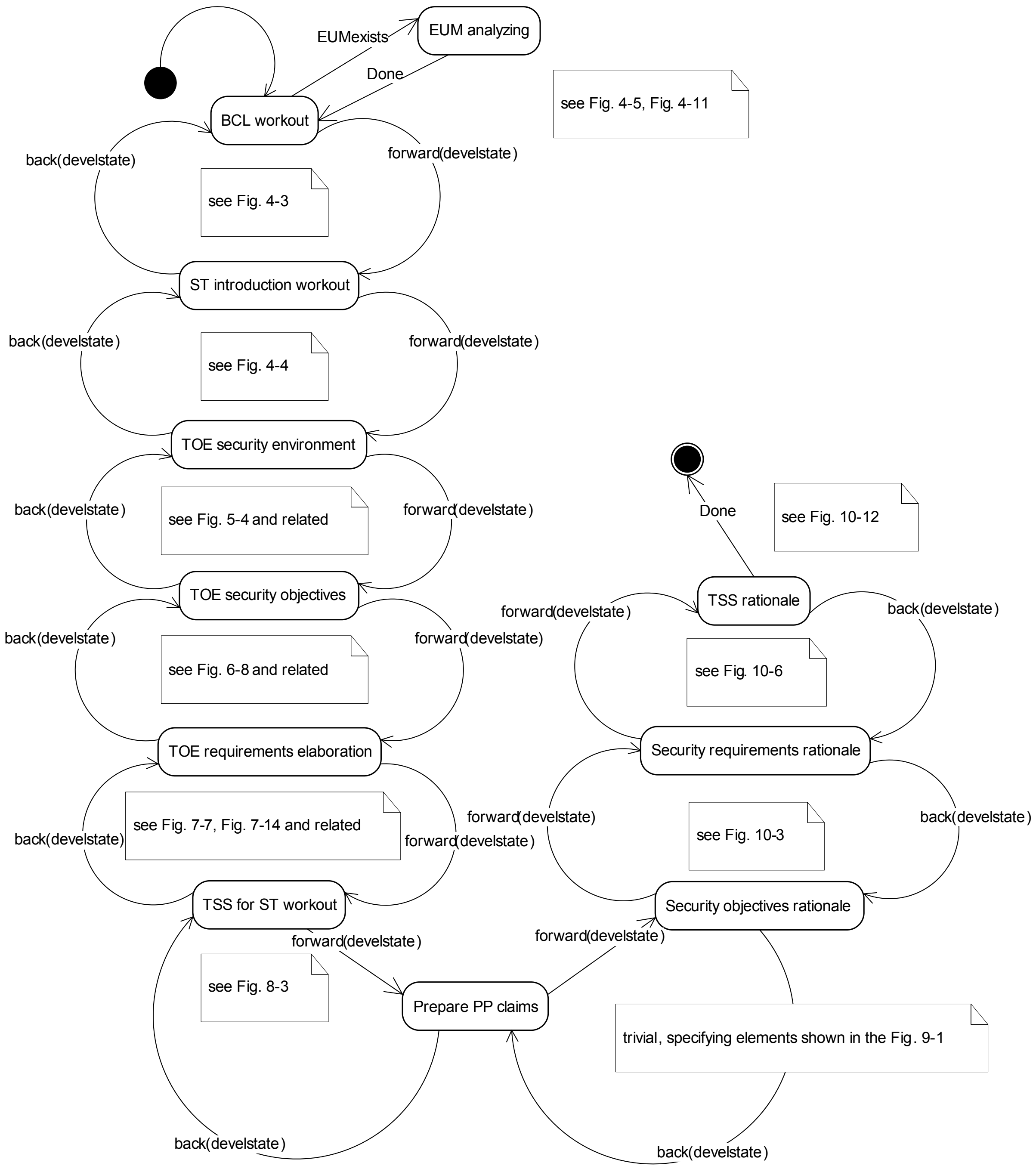


Fig. 2.6. IT security development process as the state machine
Rys. 2.6. Proces konstruowania zabezpieczeń informatycznych jako maszyna stanowa

A positive result allows to go to the next IT security development stage, while a negative result suggests passing to the current stage or to one of the prior stages to do some corrections or improvements. The name of the destination state is used as the transition parameter "*develstage*".

The entry action analyzes the received transition parameter. If it contains the entered state name, the internal state machine of this stage is put into operation and the processing is performed, going from one substate to the next, until the final substate is reached. If the parameter contains other stage name (prior or further stage), the new transition with this parameter is generated and passed to the next or previous state, regarding the ordered set of the IT security development stages.

This mechanism allows to pass freely in both directions within the ordered set of stages. The above mentioned actions have auxiliary meaning, organizing the whole of the developers' activities. They are not described directly within the Common Criteria standard, though they comply with it. The Fig. 2.6 shows only main states. Please remember that all of them have their internal state machines.

The product model (PM) elaboration will be discussed in the chapter 4. The translation of the PM model to the security model (SM) during the IT security development process will be discussed in the chapters 5 through 10, and some issues concerning the evaluation process will be presented in the chapter 11.

3. Generics, functional and assurance components – internal data representation

The chapter deals with the semiformal means considered as the "specification language" needed to build security models of the IT product or system. The accuracy of the semiformal, UML-based model can be improved by the use of the OCL [103]. This approach will also be applied to the methodology presented there. Moreover, semantic aspects will be considered by using the results of the author's earlier works. The semantics of the OCL was developed in [91], and adopted for the OCL 2.0 specification (for more details see: Appendix A "Semantics" of [83], and Appendix B of this monograph developed on this basis). Using the introduced there basic notation and definitions, the syntax and semantics of the developed IT security models will be presented.

The full compatibility with the formal OCL specification is assumed, though a specific subset of classes, attributes, and operations will be defined to better express different kinds of security issues, used by the IT security developers within the framework defined there. Especially, there is a need to present the developers' library items – predefined or user-defined, and items which are elements of any security specification.

The key OCL issue is the class $\mathsf{CLASS} \subseteq \mathcal{N}$, where $\mathcal{N}$ is a set of finite, non-empty names over the given alphabet $\mathcal{A}$. A class represents a common description for a set of objects having the same properties. Each class induces an object type $t_c \in \mathsf{T}$, having the same name as the class. Please note that CLASS represents all class names possible to define. In this monograph the subset $\mathsf{SICLASS} \subseteq \mathsf{CLASS}$ will be used to represent any security issues in the security domain. The abstract class $\mathsf{SICLASS}$ induces an object type $t_{\mathsf{SIC}} \in \mathsf{T}$. This class is a generalization of other classes defined below, representing different kinds of security issues, i.e. generics and Common Criteria components, and their taxonomy. Generally, a four-level taxonomy is assumed:

- high-level description using the defined abstract items – "generic" and "Common Criteria component",
- groups of generics and classes of components,
- families of generics and components,

- generic items and functional/assurance components – concrete items existing in the library or placed in the specifications.

All these issues will be progressively discussed and finally a formal full-class descriptor will be presented for the generic items and functional/assurance components used by the developers.

3.1. Generics as semiformal, UML-based specification means

The following general, informal definition of a generic was assumed:

Definition 3.1: Generic (informal).

A generic is a mnemonic name, expressing the set of common features, behaviours or actions, relating to different aspects or elements of an IT security system, like subjects, objects, assumptions for the security environment, organizational security policies, threats, security objectives for the TOE and its environment, security requirements for the environment, security functions, as well as vulnerabilities[4], risks, and impacts.

□

The following, dot-notation based, general format of a generic was assumed, which is very similar to the format of components and simple generics used by the developers. This notation of generics will be called the "developers' style".

Definition 3.2:[5] Syntax and semantics of a generic – a general definition.

A generic is a structure of dot-separated fields, where each optional field is marked with square brackets (a syntax):

```
Generic::=
        [domain.][group.]family.mnemonic[_Dderver][_Iinsnum].
        description.refinement[.genattrib][.genoper()].
```

The fields can be interpreted as follows (semantics):

- *domain* – specifies the area of IT applications that the generic concerns; this field allows to group the generics according to the specific application needs;
- *group* – specifies the aspects or elements of the assumed IT security model that the generic applies to; this field allows to group the generics by the security target elements;
- *family* – provides a more detailed taxonomy for the given group;
- *mnemonic* – a concise expression of a feature, behaviour, or action; may include parameters to be a generic too, usually representing assets or subjects;
- *_Dderver* – where "*_D*" is a prefix, *derver* is a successive derived version number (an identifier); for *derver*=0 the entire field can be omitted, and this means that this is a basic version of the generic; this field concerns generics placed in the library and/or security specification;

[4] This definition has a more general meaning. Generics expressing vulnerabilities, risks, impacts and other security issues are not discussed in this monograph.

[5] The main changes, in comparison with the author's earlier works: „*type*" replaced by „*family*", „*version*" by „*derver*", „*attributes*" by „*genattrib*", and „*group*", "*insnum*" and "*genoper()*") added.

- *_Iinsnum* – where "*_I*" is a prefix, *insnum* is an identifier (a successive number) of exemplars of the given generic placed in the specification; for *insnum*=0 the entire field can be omitted, and this means that only one instance (one exemplar) of the generic is placed in the specification; this field does not concern generics in the library;
- *description* – a full description, expressing *mnemonic* meaning;
- *refinement*[6] – details and interpretations dealing with the "description"; a field attached by the developer, matching the meaning of a generic to the TOE reality[7];
- *genattrib* – a list of attributes expressing additional features;
- *genoper()* – a list of internal operations expressing auxiliary aspects derived from other fields.

□

Different kinds of attributes and operations will be introduced for specific families and groups, e.g.:

- *assetValue* – the value of the asset;
- *eventLikelihood* – the likelihood of the event concerning a threat,
- *riskValueAssess()* – risk value concerning the threat;
- *preventive, corrective, detective* – Boolean flags of the security objectives.

For the *domain*, *group*, and *family* fields the appropriate enumeration types expressing the generics taxonomy are defined. i.e.: $t_{DOMAIN} \in T_E$, $t_{GROUP} \in T_E$, $t_{FAMILY} \in T_E$, while other fields are mostly $String$ type.

Definition 3.3: Syntax of generic domain.

The `Generic` field *domain* is defined as the enumeration type $t_{DOMAIN} \in T_E$, where:

$literals(t_{DOMAIN})$ = {GNR, CRP, COM, DAB, TTP, SCR, USD}.

□

Definition 3.4: Semantics of generic domain.

The semantics of enumeration type $t_{DOMAIN} \in T_E$ is a function

$I(t_{DOMAIN}) = literals(t_{DOMAIN}) \cup \{\perp\}$.

The following interpretation of literals of type t_{DOMAIN} is assumed:

$I(e_1) = I$(GNR): "Common aspects for all applications",

$I(e_2) = I$(CRP): "Specific cryptographic applications",

$I(e_3) = I$(COM): "Communication, networks aspects, firewalls, intrusion detection or prevention systems specifics",

$I(e_4) = I$(DAB): "Database Management Systems (DBMS)",

$I(e_5) = I$(TTP): "Specific applications for Trusted Third Party (TTP)",

$I(e_6) = I$(SCR): "Smart cards",

$I(e_7) = I$(USD): "User-defined domain".

□

[6] Defined analogically to the CC components refinements. Usually, the developers precede the component refinement section by the underlined word „Refinement:".

[7] Not included in the library but added to the project.

The generic can be considered as the UML class. The Fig. 3.1 shows the abstract class `Generic` $\in$ SICLASS, implying type $t_{\text{GENERIC}} \in T_{\text{SIC}}$. Please note that the fields of the generic (see Definition 3.2) can be expressed by the UML class attributes or operations.

Definition 3.5: Basic and auxiliary attributes of a generic.

Let $t \in T$ be an OCL type, N is a set of finite, non-empty names over alphabet A to be given. The attributes of the class `Generic` $\in$ SICLASS, defined as a set $\text{ATT}_{\text{Generic}}$ of signatures:

$a: t_{\text{GENERIC}} \rightarrow t$,

where the attribute name $a \in N$, and an implied class type $t_{\text{GENERIC}} \in T_{\text{SIC}}$, can be expressed as:

$\text{ATT}_{\text{Generic}} =$
$\{$*domain*: $t_{\text{GENERIC}} \rightarrow t_{\text{DOMAIN}}$,
group: $t_{\text{GENERIC}} \rightarrow t_{\text{GROUP}}$,
family: $t_{\text{GENERIC}} \rightarrow t_{\text{FAMILY}}$,
mnemonic: $t_{\text{GENERIC}} \rightarrow$ String,
derver: $t_{\text{GENERIC}} \rightarrow$ String,
insnum: $t_{\text{GENERIC}} \rightarrow$ String,
description: $t_{\text{GENERIC}} \rightarrow$ String,
refinement: $t_{\text{GENERIC}} \rightarrow$ String,
genattrib: $t_{\text{GENERIC}} \rightarrow$ OclType,
-- and auxiliary attributes for the generic management (see chapter 3.3)
userdefined: $t_{\text{GENERIC}} \rightarrow$ Boolean,
assignstat: $t_{\text{GENERIC}} \rightarrow t_{\text{ASSTAT}}\}$, where
$t_{\text{ASSTAT}} \in T_E$ and the semantics of the t_{ASSTAT} is the function
$I(t_{\text{ASSTAT}}) = \text{literals}(t_{\text{ASSTAT}}) \cup \{\perp\}$.
The following interpretation of literals of the type t_{ASSTAT} is assumed:
$I(e_1)$ = I(NON-EXISTING): “The generic or CC component is not defined yet”,
$I(e_2)$ = I(DEFINED): “The generics or CC components exist in the library”,
$I(e_3)$ = I(ASSIGNED): “Library element was added to the specification”. □

Definition 3.6: Basic operations of a generic.

Let $t \in T$ be an operation result of the OCL type, $t_1, \dots, t_n \in T$ be operation parameters of the OCL types, N is a set of finite, non-empty names over the given alphabet A. The operations of the class `Generic` $\in$ SICLASS , defined as a set $\text{OP}_{\text{Generic}}$ of signatures:

$\omega: t_{\text{GENERIC}} \times t_1 \times \dots \times t_n \rightarrow t$,

where the operation symbol $\omega \in N$, and $t_{\text{GENERIC}} \in T_{\text{SIC}}$ is an implied class type, can be expressed as:

$\text{OP}_{\text{Generic}} =$
$\{$*genoper*: $t_{\text{GENERIC}} \times$ OclType $\times$ OclType $\times \dots, \quad \times$ OclType $\rightarrow t$,
develsname: $t_{\text{GENERIC}} \times$ String $\times$ String $\times$ String $\rightarrow$ String$\}$.

□

The first symbol, “*genoper*” represents any operation defined for a generic, but the second one, the “*develsname*”, is used to identify generics objects for a given *domain* and a given *family*. The first string-type argument concerns a unique *mnemonic* of the generic, the second *derver* preceded by the prefix *_D*, and the third – the instance identifier preceded by the prefix *I_*. Besides, some other basic attributes and operations can be added to child classes in the same way.

The `Generic` class is a generalization of abstract classes representing individual domains, e.g. *`GENGeneric`*, *`COMGeneric`*, etc. Every domain generic must have the proper OCL constraint assigned concerning the *domain* attribute.

Example 3.1: OCL constraints for *domain* attribute example.[8]

```
GENGeneric
self.domain = #GEN

COMGeneric
self.domain = #COM
```
□

Expressing generics as UML classes implies another, along with the "developers' style", notation style. It will be called the "UML style", generally compliant with the UML object notation including `object name: object type`.

An additional operation is added to the generic class to issue the developers' style name, i.e. *develsname()*. It can be expressed using the OCL notation concerning pre- and post-conditions for this operation:

```
GENGeneric::develsname()
-- for generics in the library (it depends on the special
-- attribute assignstat discussed later)
pre: -- none
post: result = self.mnemonic.concat('_D'.concat(derver))
GENGeneric::develsname()
-- for generics in the specification (it depends on the
-- special attribute assignstat discussed later)
pre: -- none
post: result = self.mnemonic.concat('_D'.concat(derver('_I'.concat(insnum))))
```

□

Assuming that the `result`, containing the issued generic names that ought to be unique, is stored in the hypothetical class attribute *uniqueName:* String, i.e. `uniqueName = result`, the uniqueness of the issued generic names can be expressed with the following OCL constraint:

```
GENGeneric
self.allInstances->forAll(p1, p2 | p1<>p2 implies
p1.uniqueName <> p2.uniqueName)
```

□

Please note that the "*_Dderver*" field assures the uniqueness of the generic names in the library. The uniqueness of the generic names in the specification is assured by the additional field "*_Iinsnum*" to distinguish exemplars of the same generic, usually with different

[8] The name of class being a context of the invariant will be underlined as in [WK03].

parameters assigned. Both these fields will be used to define the function oid(`Generic`), used for the UML objects identification, where $t_{GENERIC} \in T_{SIC}$.

Ordering generics by *domain* has auxiliary meaning and is parallel to ordering them by group. Let us assume, for further considerations, that *domain* = constant, e.g. is equal to #GEN.

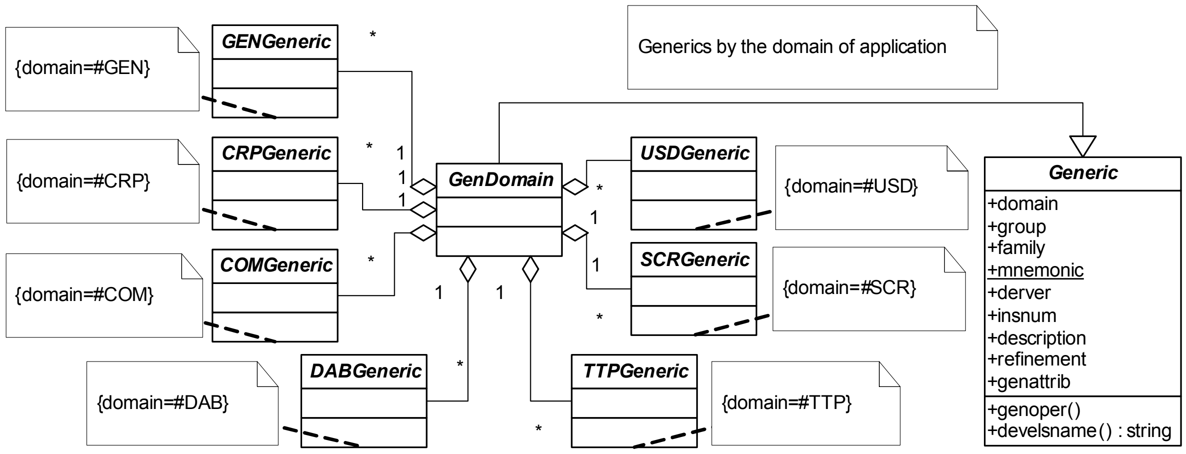


Fig. 3.1. Generic and generic domains as the UML class diagram
Rys. 3.1. Generyk jako klasa oraz domeny generyków na diagramie klas UML

Definition 3.7: Decomposition of generics with respect to the application domains.

Let $S_{Generic}$ be a set of all generics and $S_{GENGeneric}$, $S_{CRPGeneric}$, $S_{COMGeneric}$, $S_{DABGeneric}$, $S_{TTPGeneric}$, $S_{SCRGeneric}$, $S_{USDGeneric}$ be sets of generics of individual domains. The domains of generics are disjointed, which can be expressed as follows:

i. $S_{Generic} = S_{GENGeneric} \cup S_{CRPGeneric} \cup S_{COMGeneric} \cup S_{DABGeneric}$
$\cup S_{TTPGeneric} \cup S_{SCRGeneric} \cup S_{USDGeneric}$,

ii. $S_{GENGeneric} \cap S_{CRPGeneric} \cap S_{COMGeneric} \cap S_{DABGeneric}$
$\cap S_{TTPGeneric} \cap S_{SCRGeneric} \cap S_{USDGeneric} = \varnothing$.

□

Please note that the equivalent OCL-style notation of this property is:

i. $Set(t_{Generic}) = Set(t_{GENGeneric}) \cup Set(t_{CRPGeneric}) \cup Set(t_{COMGeneric}) \cup Set(t_{DABGeneric})$
$\cup Set(t_{TTPGeneric}) \cup Set(t_{SCRGeneric}) \cup Set(t_{USDGeneric})$,

ii. $Set(t_{GENGeneric}) \cap Set(t_{CRPGeneric}) \cap Set(t_{COMGeneric}) \cap Set(t_{DABGeneric})$
$\cap Set(t_{TTPGeneric}) \cap Set(t_{SCRGeneric}) \cap Set(t_{USDGeneric}) = \varnothing$.

The second, and parallel classification of generics with respect to the security model parts (i.e. security target structure) is provided by the attribute *group*.

Definition 3.8: Syntax of generic group.

The `Generic` field *group* is defined as the enumeration type $t_{GROUP} \in T_E$, where:
literals(t_{GROUP}) = {DAgr, Sgr, Tgr, Pgr, Agr, Ogr, REgr, Fgr}.

□

Definition 3.9: Semantics of generic group.

The semantics of enumeration type $t_{\mathrm{GROUP}} \in T_E$ is a function

$$I(t_{\mathrm{GROUP}}) = \mathrm{literals}(t_{\mathrm{GROUP}}) \cup \{\perp\}.$$

The following interpretation of literals of type t_{GROUP} is assumed:

$I(e_1) = I(\mathrm{DAgr})$: "Assets, passive entities – internal or external to the TOE",

$I(e_2) = I(\mathrm{Sgr})$: "Subjects, active entities – legal or illegal, sources of undesirable events",

$I(e_3) = I(\mathrm{Tgr})$: "Threats concerning the TOE and its environment",

$I(e_4) = I(\mathrm{Pgr})$: "OSP (Organizational Security Policies) rules",

$I(e_5) = I(\mathrm{Agr})$: "Assumptions",

$I(e_6) = I(\mathrm{Ogr})$: "Security objectives for the TOE and its environment",

$I(e_7) = I(\mathrm{REgr})$: "Security requirements for the TOE environment",

$I(e_8) = I(\mathrm{Fgr})$: "Trusted security functions".

□

The Fig. 3.2 shows abstract class `Generic` ∈ SiClass as a generalization of generics classes representing particular groups. Please note specific attributes added for some classes and OCL constraints concerning the class attribute *group* that can be specified in the similar way as in the example 3.1. Some types of generics have *paramDAgr*/*paramSgr* attributes assigned, allowing the parameterization of generics which is discussed later.

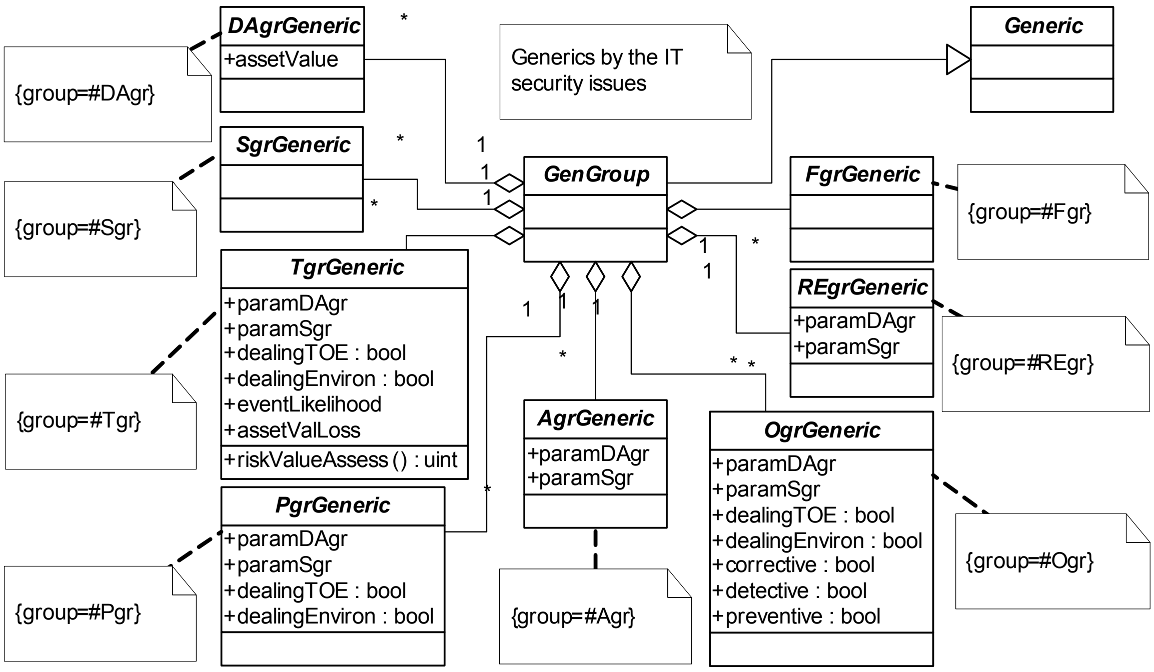


Fig. 3.2. Generic groups
Rys. 3.2. Grupy generyków

Definition 3.10: Decomposition of generics with respect to the generic groups.

Let $S_{Generic}$ be a set of all generics, and $S_{DAgrGeneric}$, $S_{SgrGeneric}$, $S_{TgrGeneric}$, $S_{PgrGeneric}$, $S_{AgrGeneric}$, $S_{OgrGeneric}$, $S_{REgrGeneric}$, $S_{FgrGeneric}$ be sets of generics of the individual groups. The groups of generics are disjointed, which can be expressed as follows:

i. $S_{Generic} = S_{DAgrGeneric} \cup S_{SgrGeneric} \cup S_{TgrGeneric} \cup S_{PgrGeneric} \cup S_{AgrGeneric} \cup S_{OgrGeneric} \cup S_{REgrGeneric} \cup S_{FgrGeneric}$,

ii. $S_{DAgrGeneric} \cap S_{SgrGeneric} \cap S_{TgrGeneric} \cap S_{PgrGeneric} \cap S_{AgrGeneric} \cap S_{OgrGeneric} \cap S_{REgrGeneric} \cap S_{FgrGeneric} = \emptyset$

□

The next generic attribute (field) – see Definition 3.5 – is the *family* field. It represents the set of different enumeration types dedicated to the individual group of generics:

i. $Set(t_{FAMILY}) = Set(t_{DAgrfam}) \cup Set(t_{Sgrfam}) \cup Set(t_{Tgrfam}) \cup Set(t_{Pgrfam}) \cup Set(t_{Agrfam}) \cup Set(t_{Ogrfam}) \cup Set(t_{REgrfam}) \cup Set(t_{Fgr})$,

ii. $Set(t_{DAgrfam}) \cap Set(t_{Sgrfam}) \cap Set(t_{Tgrfam}) \cap Set(t_{Pgrfam}) \cap Set(t_{Agrfam}) \cap Set(t_{Ogrfam}) \cap Set(t_{REgrfam}) \cap Set(t_{Fgr}) = \emptyset$,

and t_{FAMILY}. $\in T_E$. For each group a set of families is defined that ought to be considered separately. The real library or specification elements are represented by the particular family members, i.e. generic items.

Definition 3.11: Syntax of generic families of the group dealing with the assets.

If the `Generic` field *group = #DAgr*, the generic field *family* is defined as the enumeration type $t_{DAgrfam} \in T_E$, where:

$literals(t_{DAgrfam}) = \{DAD, DAS, DAE, DAP\}$

and the set of attributes is:

$ATT_{DAgrGeneric} = \{assetValue\}$,

and the set of operations is:

$OP_{DAgrGeneric} = \emptyset$.

□

Definition 3.12: Semantics of generic families of the group dealing with the assets.

The semantics of enumeration type $t_{DAgrfam} \in T_E$ is a function

$I(t_{DAgrfam}) = literals(t_{DAgrfam}) \cup \{\perp\}$.

The following interpretation of literals of type $t_{DAgrfam}$ is assumed:

$I(e_1) = I(DAD)$: “Data objects and other assets”,

$I(e_2) = I(DAS)$: “Asset as service”,

$I(e_3) = I(DAE)$: “General purpose assets placed in the TOE IT environment”,

$I(e_4) = I(DAP)$: “General purpose assets placed in physical environment of the TOE”.

□

The Fig. 3.3 shows the abstract class `DAgrGeneric` $\in$ SiClass as a generalization of generics classes representing its particular families. The diagram shows a four-level model of data. The generic items belonging to different families of particular groups represent library items used to create security specifications.

Definition 3.13: Decomposition of `DAgrGeneric` with respect to its families.

Let $S_{DAgrGeneric}$ be a set of all generics with *group = #DAgr*, and S_{DADGen}, S_{DASGen}, S_{DAEGen}, S_{DAPGen} be sets of generics of individual families. The families of `DAgrGenerics` are disjointed, which can be expressed as follows:

i. $S_{DAgrGeneric} = S_{DADGen} \cup S_{DASGen} \cup S_{DAEGen} \cup S_{DAPGen}$,

ii. $S_{DADGen} \cap S_{DASGen} \cap S_{DAEGen} \cap S_{DAPGen} = \varnothing$.

□

Please note that each of the S_{DADGen} S_{DASGen} S_{DAEGen} S_{DAPGen} sets of generics contains its own elements, i.e. `DADItem` $\in S_{DADGen}$, `DASItem` $\in S_{DASGen}$, `DAEItem` $\in S_{DAEGen}$ and `DAPItem` $\in S_{DAPGen}$. For example:

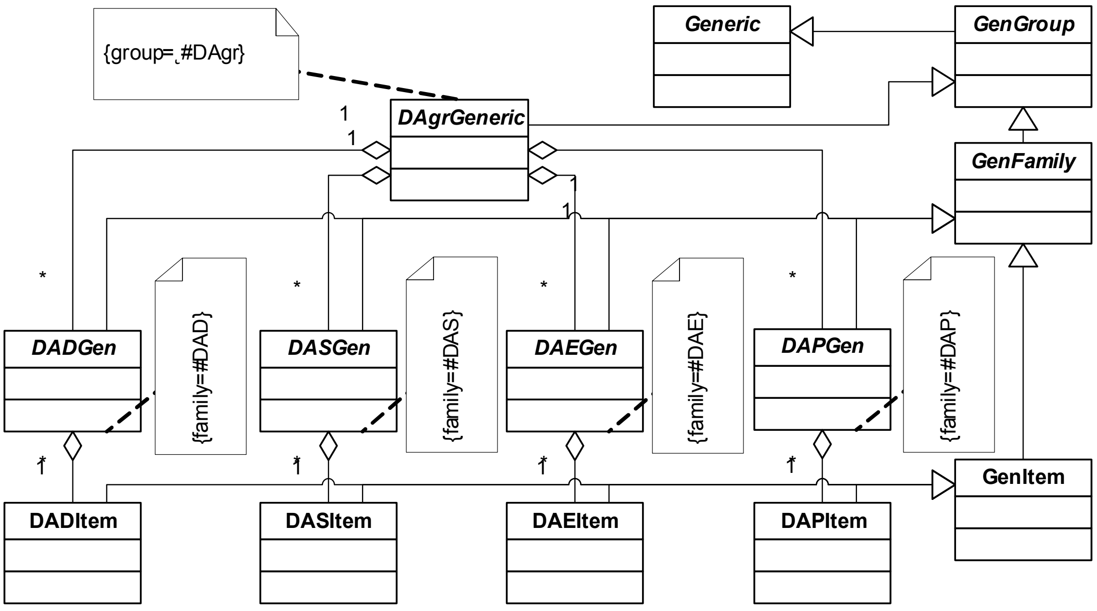


Fig. 3.3. Generic families expressing data and other assets
Rys. 3.3. Rodziny generyków reprezentujące dane i inne zasoby

- S_{DADGen} = {BioData, CertStore, CipherText, EncKey, HashVal, InputData, OutputData, PlainText, ...},
- S_{DASGen} = {CryptoAPIServices, SysServices,...}.

The `GenItem` elements represent the concrete generics of the given family placed in the library or defined by the developers. The generic family can be decomposed to generics, and the semantics of generics represented by their description can be defined in the same way as the groups were decomposed to the families.

All generic items (`GenItem`), belonging to different families, must have distinct names. For this reason, the class attribute *mnemonic* (Fig. 3.1) has the `scope=classifier`, and is marked by underlining. It is the key issue in constructing unique generics names (see *develsname()* operation), used as the objects identifiers and role names for the navigation through the model.

Due to large number of generics, the developed UML model is a bit simplified and abstract classes are used. Each of them is expressed by `GenItem` or a given family item, e.g. `DADItem`, `DASItem`, `OCONItem`, `TDAItem` to express common features or behaviour. Only examples of generics can be shown there, and the full list of generics is placed in the technical documentation [94]. This remark concerns any generic family defined further in this chapter. Similar simplifications, allowed in the UML modelling, will be introduced later for the Common Criteria components.

Definition 3.14: Syntax of generic families of the group dealing with the subjects.

If the `Generic` field *group* = *#Sgr*, the generic field *family* is defined as the enumeration type $t_{Sgrfam} \in T_E$, where:

$literals(t_{Sgrfam})$ = {SNA, SAU, SAH, SNH}

and the set of attributes is:

$ATT_{SgrGeneric} = \varnothing$,

and the set of operations is:

$OP_{SgrGeneric} = \varnothing$.

□

Definition 3.15: Semantics of generic families of the group dealing with the subjects.

The semantics of enumeration type $t_{Sgrfam} \in T_E$ is a function

$I(t_{Sgrfam}) = literals(t_{Sgrfam}) \cup \{\perp\}$.

The following interpretation of literals of type t_{Sgrfam} is assumed:

$I(e_1)$ = I(SNA): "Represents an unauthorized subject (individual, user, process); may be internal or external to the TOE; usually expresses threat agents",

$I(e_2)$ = I(SAU): "Represents an authorized subject; may be internal or external to the TOE; usually expresses legal users or administrators",

$I(e_3)$ = I(SAH): "Deals with the source of an undesirable event caused by accidental human actions or errors",

$I(e_4)$ = I(SNH): "Deals with the source of an undesirable event caused by non-human actions, deals with physical environment, like fire, flood, earthquake, different disturbances or technical failures".

□

The Fig. 3.4 shows the abstract class `SgrGeneric` ∈ SICLASS as a generalization of generics classes representing its particular families and their items.

Definition 3.16: Decomposition of `SgrGeneric` with respect to its families.

Let $S_{SgrGeneric}$ be a set of all generics with *group* = *#Sgr*, and S_{SNAGen}, S_{SAUGen}, S_{SAHGen}, S_{SNHGen} be sets of generics of individual families. The families of `SgrGeneric` are disjointed, which can be expressed as follows:

i. $S_{SgrGeneric} = S_{SNAGen} \cup S_{SAUGen} \cup S_{SAHGen} \cup S_{SNHGen}$,

ii. $S_{SNAGen} \cap S_{SAUGen} \cap S_{SAHGen} \cap S_{SNHGen} = \varnothing$.

□

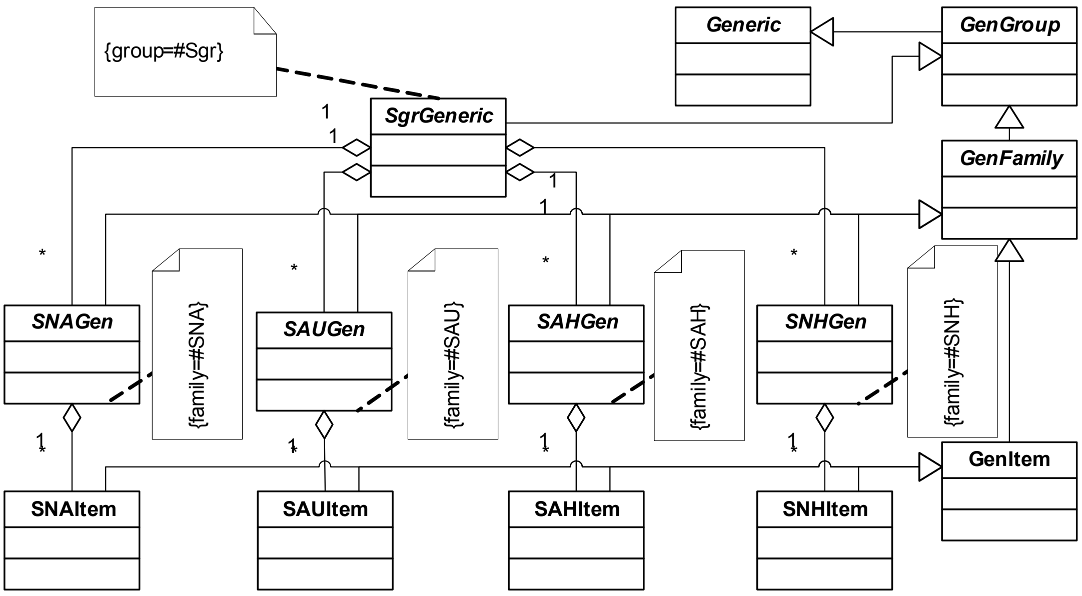


Fig. 3.4. Subject generic families
Rys. 3.4. Rodziny reprezentujące podmioty

Definition 3.17: Syntax of generic families of the group dealing with the threats.

If the `Generic` field *group = #Tgr*, the generic field *family* is defined as the enumeration type $t_{Tgrfam} \in T_E$, where:

$literals(t_{Tgrfam})$ = {TDA, TUA, TAA, TIT, TPH, TFM}

and the set of attributes is:

$ATT_{TgrGeneric}$ = {*paramDAgr, paramSgr, dealingTOE, dealingEnviron, eventLikelihood, assetValLoss*},

and the set of operations is:

$OP_{TgrGeneric}$ = {*riskValueAssess*}.

□

Definition 3.18: Semantics of generic families of the group dealing with the threats.

The semantics of enumeration type $t_{Tgrfam} \in T_E$ is a function

$I(t_{Tgrfam}) = literals(t_{Tgrfam}) \cup \{\perp\}$.

The following interpretation of literals of type t_{Tgrfam} is assumed:

$I(e_1) = I(TDA)$: "Concerns direct attacks made by hackers and other intruders, including users and administrators",

$I(e_2) = I(TUA)$: "Deals with users' activities, except their direct, malicious activities",

$I(e_3) = I(TAA)$: "Concerns administrators' activities, except their direct, malicious activities",

$I(e_4) = I(TIT)$: "Deals with IT aspects – software (flaws, malicious codes, etc.) or hardware (failures, power disruption, tampering, electromagnetic emanation, etc.)",

$I(e_5) = I(TPH)$: "Deals with technical infrastructure and physical security of the TOE environment",

$I(e_6) = I(TFM)$: "Concerns force majeures, accidents, catastrophes, terrorism acts, and other undesired events, and failures possible within the TOE environment". □

More detailed information concerning semantics of the threat families is included in the Table 3.1 below. The Fig. 3.5 shows the abstract class `TgrGeneric` ∈ SICLASS as a generalization of generics classes representing its particular families and their items.

Definition 3.19: Decomposition of `TgrGeneric` with respect to its families.

Let $S_{TgrGeneric}$ be a set of all generics with *group* = *#Tgr*, and S_{TDAGen}, S_{TUAGen}, S_{TAAGen}, S_{TITGen}, S_{TPHGen}, S_{TFMGen} be sets of generics of individual families. The families of `TgrGeneric` are disjointed, which can be expressed as follows:

i. $S_{TgrGeneric} = S_{TDAGen} \cup S_{TUAGen} \cup S_{TAAGen} \cup S_{TITGen} \cup S_{TPHGen} \cup S_{TFMGen}$,

ii. $S_{TDAGen} \cap S_{TUAGen} \cap S_{TAAGen} \cap S_{TITGen} \cap S_{TPHGen} \cap S_{TFMGen} = \varnothing$.

□

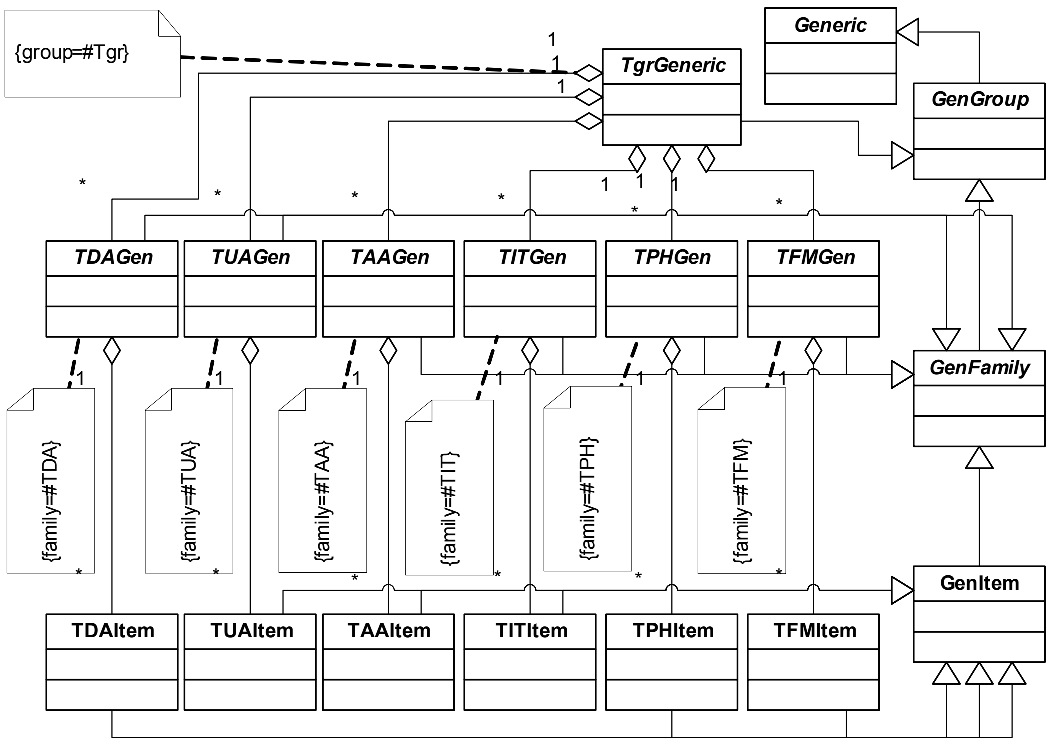


Fig. 3.5. Generic families dealing with threats
Rys. 3.5. Rodziny generyków dotyczące zagrożeń

Definition 3.20: Syntax of generic families of the group dealing with the security policy rules.

If the `Generic` field *group* = *#Pgr*, the generic field *family* is defined as the enumeration type $t_{Pgrfam} \in T_E$, where:

literals(t_{Pgrfam}) = {PIDA, PACC, PADT, PINT, PAVB, PPRV, PDEX, PCON, PEIT, PEPH, PSMN, POTL},

and the set of attributes is:

$ATT_{PgrGeneric}$ = {*paramDAgr, paramSgr, dealingTOE, dealingEnviron*},

and the set of operations is:

$OP_{PgrGeneric} = \varnothing$.

□

Definition 3.21: Semantics of generic families of the group concerning the security policy rules.

The semantics of enumeration type $t_{Pgrfam} \in T_E$ is a function

$I(t_{Pgrfam}) = literals(t_{Pgrfam}) \cup \{\perp\}$.

The following interpretation of literals of type t_{Pgrfam} is assumed:

$I(e_1) = I(PIDA)$: "Deals with identification and authentication",
$I(e_2) = I(PACC)$: "Specifies access control and information flow control rules",
$I(e_3) = I(PADT)$: "Concerns accountability and security audit",
$I(e_4) = I(PINT)$: "Concerns integrity",
$I(e_5) = I(PAVB)$: "Concerns availability",
$I(e_6) = I(PPRV)$: "Deals with privacy",
$I(e_7) = I(PDEX)$: "Specifies general secure data exchange rules",
$I(e_8) = I(PCON)$: "Deals with confidentiality",
$I(e_9) = I(PEIT)$: "Deals with the right use of software and hardware within the TOE environment",
$I(e_{10}) = I(PEPH)$: "Deals with technical infrastructure (media) and physical security of the TOE environment",
$I(e_{11}) = I(PSMN)$: "Encompasses security maintenance (management) aspects",
$I(e_{12}) = I(POTL)$: "Concerns technical solutions and legislation, obligatorily used within the organization".

□

More detailed information concerning semantics of the OSP families is included in the Table 3.2 below. The Fig. 3.6 shows the abstract class `PgrGeneric` $\in$ SICLASS as a generalization of generics classes representing its individual families with their items.

Definition 3.22: Decomposition of `PgrGeneric` with respect to its families.

Let $S_{PgrGeneric}$ be a set of all generics with *group* = *#Pgr*, and $S_{PIDAGen}$, $S_{PACCGen}$, $S_{PADTGen}$, $S_{PINTGen}$, $S_{PAVBGen}$, $S_{PPRVGen}$, $S_{PDEXGen}$, $S_{PCONGen}$, $S_{PEITGen}$, $S_{PEPHGen}$, $S_{PSMNGen}$, $S_{POTLGen}$ be sets of generics of individual families. The families of `PgrGeneric` are disjointed, which can be expressed as follows:

i. $S_{PgrGeneric} = S_{PIDAGen} \cup S_{PACCGen} \cup S_{PADTGen} \cup S_{PINTGen} \cup S_{PAVBGen} \cup S_{PPRVGen} \cup S_{PDEXGen} \cup S_{PCONGen} \cup S_{PEITGen} \cup S_{PEPHGen} \cup S_{PSMNGen} \cup S_{POTLGen}$,

ii. $S_{PIDAGen} \cap S_{PACCGen} \cap S_{PADTGen} \cap S_{PINTGen} \cap S_{PAVBGen} \cap S_{PPRVGen} \cap S_{PDEXGen} \cap S_{PCONGen} \cap S_{PEITGen} \cap S_{PEPHGen} \cap S_{PSMNGen} \cap S_{POTLGen} = \varnothing$.

□

Definition 3.23: Syntax of generic families of the group dealing with the assumptions for the environment.

If the `Generic` field *group* = *#Agr*, the generic field *family* is defined as the enumeration type $t_{Agrfam} \in T_E$, where:

$literals(t_{Agrfam}) = \{AX, AU, AE, AC, AP, AA\}$

and the set of attributes is:

$ATT_{AgrGeneric} = \{paramDAgr, paramSgr\}$, and the set of operations is:

$\mathrm{Op}_{\mathrm{AgrGeneric}} = \varnothing$.

□

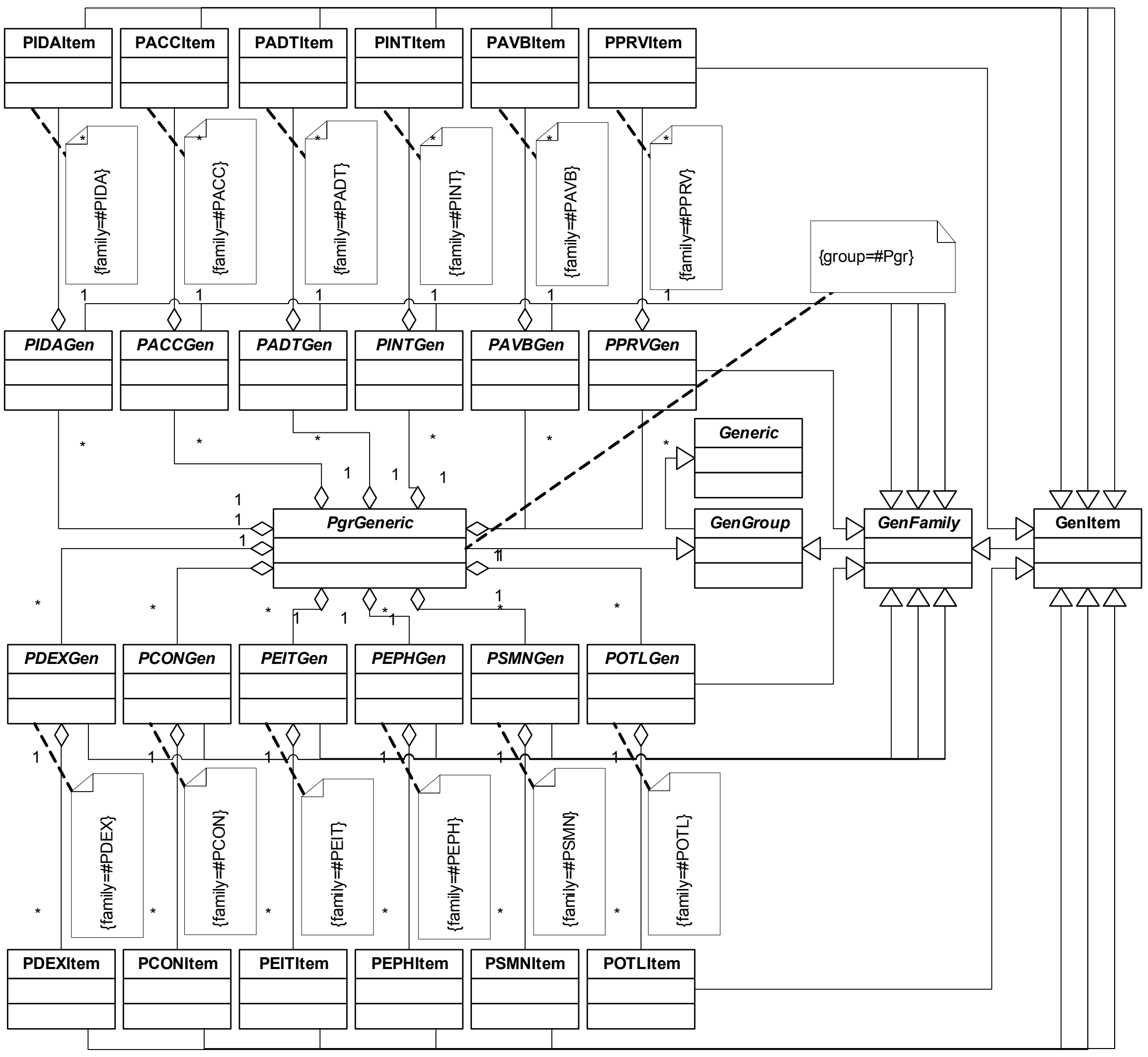


Fig. 3.6. Generic families dealing with OSP rules

Rys. 3.6. Rodziny generyków reprezentujące reguły polityki bezpieczeństwa

Definition 3.24: Semantics of generic families of the group dealing with the assumptions for the environment.

The semantics of enumeration type $t_{\mathrm{Agrfam}} \in T_E$ is a function

$I(t_{\mathrm{Agrfam}}) = \mathrm{literals}(t_{\mathrm{Agrfam}}) \cup \{\perp\}$.

The following interpretation of literals of type t_{Agrfam} is assumed:

$I(e_1) = I(\mathrm{AX})$: "Deals with the relevance of the considered threat",

$I(e_2) = I(\mathrm{AU})$: "Deals with the intended usage of the TOE",

$I(e_3) = I(\mathrm{AE})$: "Must be satisfied by the TOE environment (i.e. in a technical or physical way)",

$I(e_4) = I(\mathrm{AC})$: "Deals with the connectivity aspects of the TOE",

$I(e_5) = I(\mathrm{AP})$: "Deals with the personnel",

$\mathfrak{l}(e_6) = \mathfrak{l}(AA)$: “Leads to a choice-given assurance requirement”.

□

The Fig. 3.7 shows the abstract class `AgrGeneric` ∈ SICLASS as a generalization of generics classes representing each of family of this group.

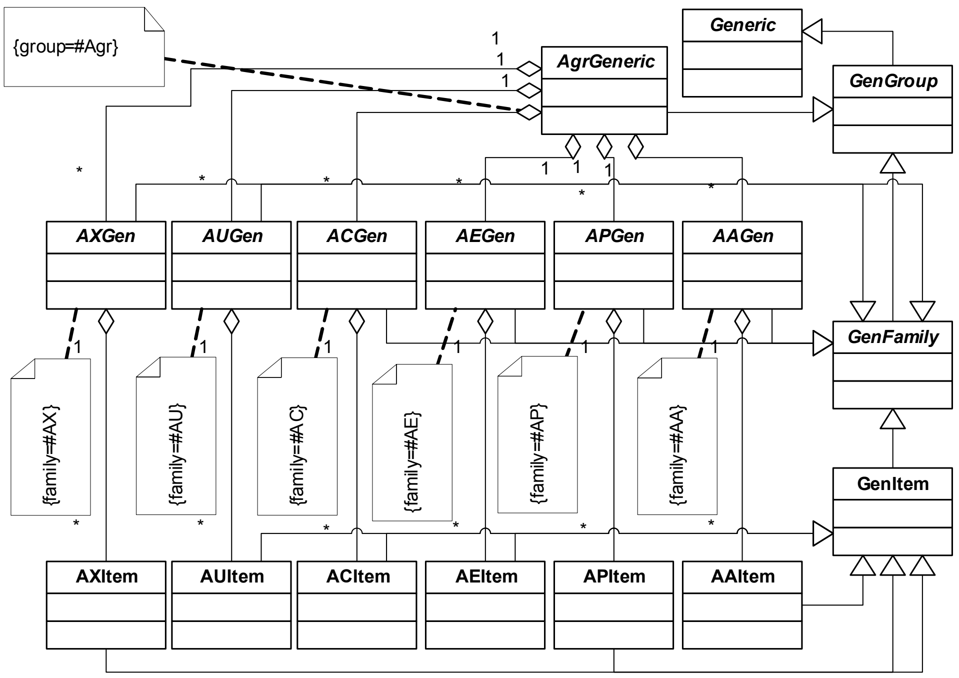


Fig. 3.7. Generic families dealing with the assumptions
Rys. 3.7. Rodziny generyków dotyczące założeń dla otoczenia zabezpieczeń

Definition 3.25: Decomposition of `AgrGeneric` with respect to its families.

Let $S_{AgrGeneric}$ be a set of all generics with *group* = *#Agr*, and S_{AXGen}, S_{AUGen}, S_{ACGen}, S_{AEGen}, S_{APGen}, S_{AAGen} be sets of generics of the individual families. The families of `AgrGeneric` are disjointed, which can be expressed as follows:
i. $S_{AgrGeneric} = S_{AXGen} \cup S_{AUGen} \cup S_{ACGen} \cup S_{AEGen} \cup S_{APGen} \cup S_{AAGen}$,
ii. $S_{AXGen} \cap S_{AUGen} \cap S_{ACGen} \cap S_{AEGen} \cap S_{APGen} \cap S_{AAGen} = \varnothing$.

□

The Fig. 3.8 shows the abstract class `OgrGeneric` ∈ SICLASS as a generalization of generics classes representing its individual families and their items. Please note that the OCL constraints regarding *family* class attribute for this group and also that the `OEITGen`, `OEPHGen`, `OSMNGen` concern only the environment.

Definition 3.26: Syntax of generic families of the group dealing with the IT security objectives for the TOE or its IT environment.

If the `Generic` field *group* = *#Ogr*, the generic field *family* is defined as the enumeration type $t_{Ogrfam} \in T_E$, where:

$\mathfrak{literals}(t_{Ogrfam})$ = {OIDA, OACC, OADT, OINT, OAVB, OPRV, ODEX, OCON, OEIT, OEPH, OSMN}

and the set of attributes is:

$\mathcal{A}\mathrm{TT}_{\mathrm{OgrGeneric}}$ = {*paramDAgr, paramSgr, dealingTOE, dealingEnviron, corrective, detective, preventive*},
and the set of operations is:
$\mathrm{OP}_{\mathrm{OgrGeneric}} = \varnothing$.
□

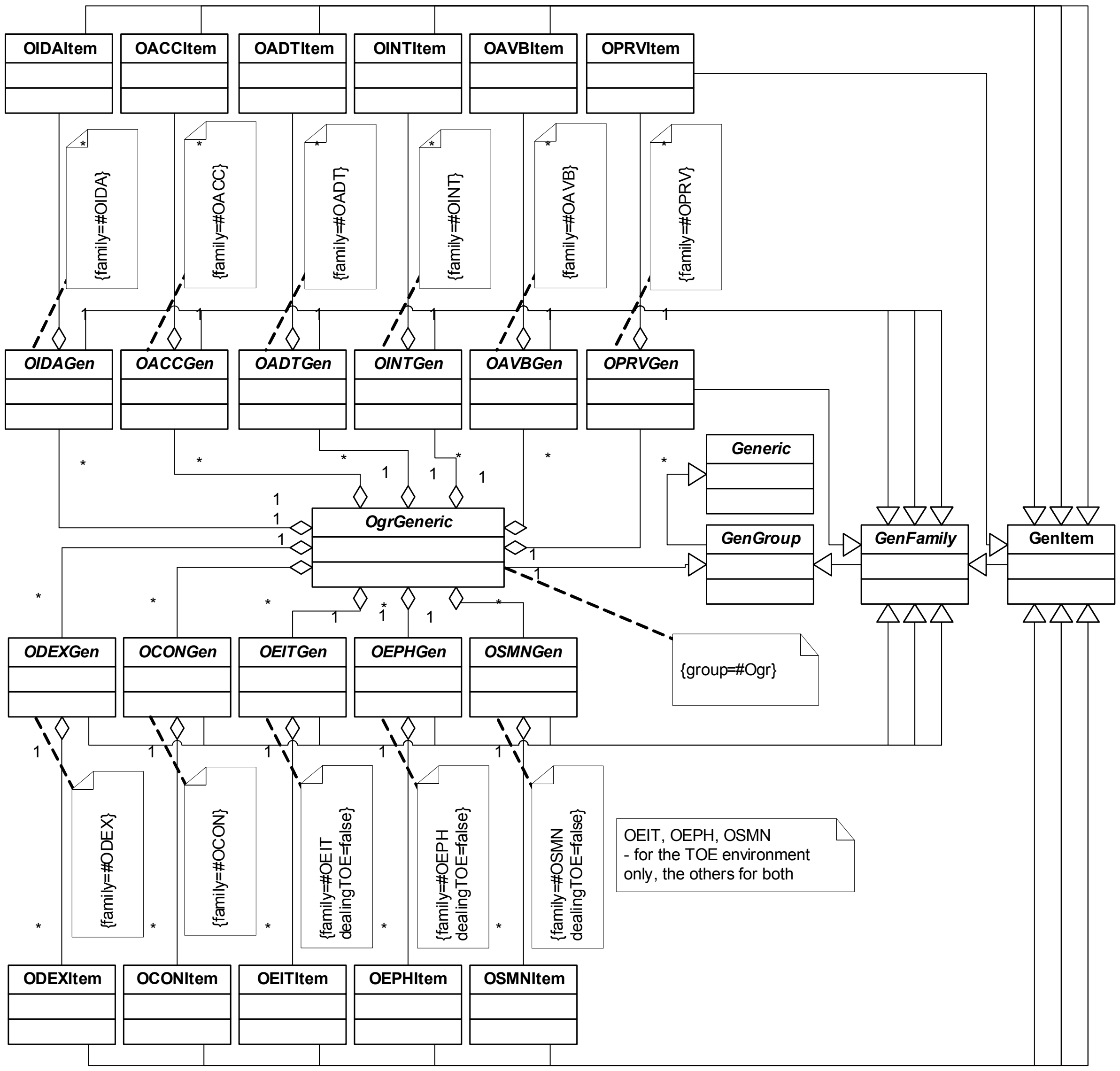


Fig. 3.8. Generic families dealing with the security objectives
Rys. 3.8. Rodziny generyków dotyczące celów zabezpieczeń

Definition 3.27: Semantics of generic families of the group dealing with the IT security objectives for the TOE or its IT environment.

The semantics of enumeration type $t_{\mathrm{Ogrfam}} \in T_E$ is a function
$I(t_{\mathrm{Ogrfam}}) = \mathrm{literals}(t_{\mathrm{Ogrfam}}) \cup \{\perp\}$.
The following interpretation of literals of type t_{Ogrfam} is assumed:
$I(e_1) = I(\mathrm{OIDA})$: “Deals with identification or authentication”,
$I(e_2) = I(\mathrm{OACC})$: “Deals with access control and information flow control rules”,

$I(e_3) = I(\text{OADT})$: “Concerns accountability and security audit”,
$I(e_4) = I(\text{OINT})$: “Concerns integrity”,
$I(e_5) = I(\text{OAVB})$: “Concerns availability”,
$I(e_6) = I(\text{OPRV})$: “Deals with privacy”,
$I(e_7) = I(\text{ODEX})$: “Concerns data exchange”,
$I(e_8) = I(\text{OCON})$: “Deals with confidentiality”,
$I(e_9) = I(\text{OEIT})$: “Deals with software or hardware aspects of the TOE environment”,
$I(e_{10}) = I(\text{OEPH})$: “Deals with technical infrastructure and physical security of the TOE environment”,
$I(e_{11}) = I(\text{OSMN})$: “Deals with security maintenance (management) – all non-IT aspects”.

□

Definition 3.28: Decomposition of `OgrGeneric` with respect to its families.

Let $S_{OgrGeneric}$ be a set of all generics with *group* = *#Ogr*, and $S_{OIDAGen}$, $S_{OACCGen}$, $S_{OADTGen}$, $S_{OINTGen}$, $S_{OAVBGen}$, $S_{OPRVGen}$, $S_{ODEXGen}$, $S_{OCONGen}$, $S_{OEITGen}$, $S_{OEPHGen}$, $S_{OSMNGen}$ be sets of generics of individual families. The families of `OgrGeneric` are disjointed, which can be expressed as follows:

i. $S_{OgrGeneric} = S_{OIDAGen} \cup S_{OACCGen} \cup S_{OADTGen} \cup S_{OINTGen} \cup S_{OAVBGen} \cup S_{OPRVGen} \cup S_{ODEXGen} \cup S_{OCONGen} \cup S_{OEITGen} \cup S_{OEPHGen} \cup S_{OSMNGen}$,

ii. $S_{OIDAGen} \cap S_{OACCGen} \cap S_{OADTGen} \cap S_{OINTGen} \cap S_{OAVBGen} \cap S_{OPRVGen} \cap S_{ODEXGen} \cap S_{OCONGen} \cap S_{OEITGen} \cap S_{OEPHGen} \cap S_{OSMNGen} = \varnothing$.

□

Definition 3.29: Syntax of generic families of the group dealing with the security requirements for the environment, impossible or difficult to express by functional or assurance components.

If the `Generic` field *group* = *#REgr*, the generic field *family* is defined as the enumeration type $t_{REgrfam} \in T_E$, where:

$literals(t_{REgrfam}) = \{\text{REIT, REPH, RENIT}\}$,
and the set of attributes is:
$ATT_{REgrGeneric} = \{paramDAgr, paramSgr\}$,
and the set of operations is:
$Op_{REgrGeneric} = \varnothing$.

□

Definition 3.30: Semantics of generic families of the group dealing with the security requirements for the environment, impossible or difficult to express by functional or assurance components.

The semantics of enumeration type $t_{REgrfam} \in T_E$ is a function
$I(t_{REgrfam}) = literals(t_{REgrfam}) \cup \{\perp\}$.
The following interpretation of literals of type $t_{REgrfam}$ is assumed:
$I(e_1) = I(\text{REIT})$: “Security requirements for the environment – general IT aspects, difficult to express with the use of functional components”,

$I(e_2)$ = I(REPH): “Security requirements for the environment dealing with technical infrastructure or physical security”,

$I(e_3)$ = I(RENIT): “Non-IT security requirements for the environment – difficult to express with the use of assurance components”.

□

The Fig. 3.9 shows the abstract class `REgrGeneric` $\in$ SICLASS as a generalization of generics classes representing its particular families. Please note the OCL constraints regarding *family* and *dealingTOE* class attributes.

Definition 3.31: Decomposition of `REgrGeneric` with respect to its families.

Let $S_{REgrGeneric}$ be a set of all generics with *group* = *#REgr*, and $S_{REITGen}$, $S_{REPHGen}$, $S_{RENITGen}$ be sets of generics of individual families. The families of `REgrGeneric` are disjointed, which can be expressed as follows:

i. $S_{REgrGeneric} = S_{REITGen} \cup S_{REPHGen} \cup S_{RENITGen}$,

ii. $S_{REITGen} \cap S_{REPHGen} \cap S_{RENITGen} = \varnothing$.

□

Definition 3.32: Syntax of generic family of the group dealing with the security functions.

If the `Generic` field *group* = *#Fgr*, the generic field *family* is defined as the enumeration type $t_{Fgrfam} \in T_E$, where:

$literals(t_{Fgrfam}) = \{F\}$,

and the set of attributes is:

$ATT_{FgrGeneric} = \varnothing$,

and the set of operations is:

$OP_{FgrGeneric} = \varnothing$. □

Definition 3.33: Semantics of generic family of the group dealing with the security functions.

The semantics of enumeration type $t_{Fgrfam} \in T_E$ is a function

$I(t_{Fgrfam}) = literals(t_{Fgrfam}) \cup \{\perp\}$.

The following interpretation of literals of type t_{Fgrfam} is assumed:

$I(e_1)$ = I(F): “Security functions derived from the functional security requirements for the TOE, called “trusted security functions – TSF”.

□

The Fig. 3.9 shows also the abstract class `FgrGeneric` $\in$ SICLASS being a generalization of the FGen class. Please note the OCL constraints regarding *family* and *dealingTOE* class attributes. Taxonomy of this group requires further investigation, for this reason only one family is defined, and its decomposition is trivial.

Definition 3.34: Decomposition of `FgrGeneric` with respect to its families.

Let $S_{FgrGeneric}$ be a set of all generics with *group* = *#Fgr*, and S_{FGen} be a set of generics of this family. The group `FgrGeneric` contains only one family FGen:

$S_{FgrGeneric} = S_{FGen}$. □

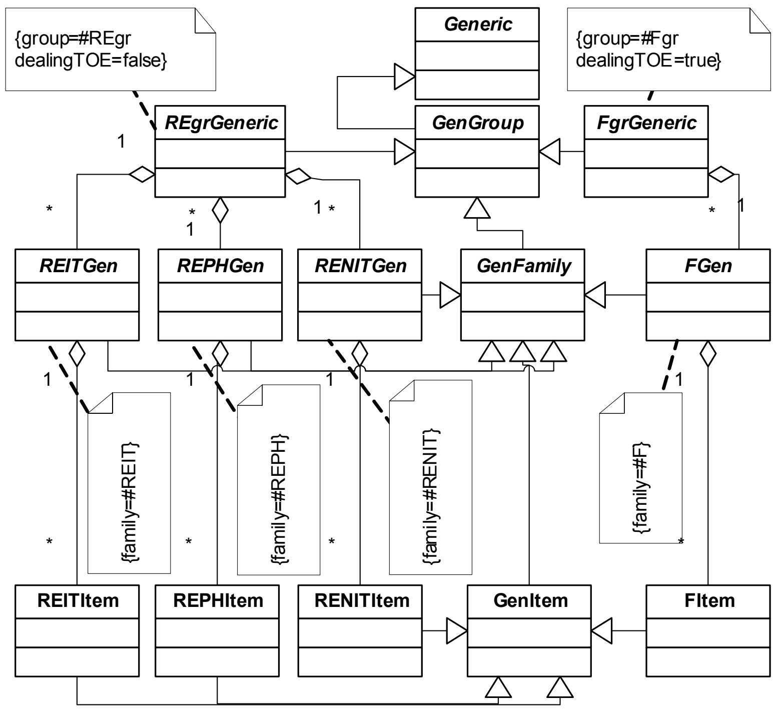


Fig. 3.9. Generic families dealing with the security requirement for the environment and dealing with the security functions
Rys. 3.9. Rodziny generyków dotyczące wymagań na zabezpieczenia środowiska oraz dotyczące funkcji zabezpieczających

The assumed definition of a generic has open character – a new application area or new security aspects can be added. It shows common rules used to build a generic set implemented within the application library. It would be very difficult to specify all possible mnemonics due to their open nature. Thus only general rules will be shown in this example. Please note that families contain generic items existing in the library and used to build security specifications.

Example 3.2: Simple and derived generics[9] placed in the developer's library.

DAD.StoredData. *Data stored on media*
DAD.StoredData_D1. *Data stored on flash memory*
The latter one is a derived (more compliant with the designer's needs) version of the above mentioned.
SNA.CleaningPers. *Internal personnel, not authorized to access the server room*
OCON.DataEncrypt. *Use data encryption*
F.DataIntegrityCtrl. *Data integrity control function (software or hardware module representation).*
□

To allow recognition and formal specification of the above mentioned different kinds of elementary security issues, called generics, the following grammar is proposed.

[9] All generics descriptions are shortened in comparison with the descriptions in real projects.

Definition 3.35: Grammar of generics.

Let us assume:

A_N – set of non-terminal symbols,
A_T – set of terminal symbols,
P – set of production,
STS – starting symbol of grammar,

The following structure is called Grammar of generics G_G :

$G_G = (A_N, A_T, P, STS)$, where:

A_N ::= <`Generic`>,<*domain*>,<*group*>,<*family*>,<*mnemonic*>,<*derver*>,<*insnum*>, <*description*>, <*refinement*>,<GenAttribOp>,<Prefix>,<BasicGen>,<Postfix>, <MnemParameter>, <RefmPrefix>,<DerVerPrefix>,<InsNumPrefix>, <AttrPrefix>,<AttrList>,<AttrItem><AttrName>,<AttrValue>,

A_T ::= [,], ., <=, (,), :,

-- [,] are used for the specification of the parameter being a generic
-- . (a dot) is used to separate different parts of a generic
-- <= means substitution of value to a parameter or attribute
-- () is used to distinguish a generic operation from generic attributes
-- : separates an attribute, operation or argument name from its type

GNR, CRP, COM, DAB, TTP, SCR, USD
DAgr, Sgr, Tgr, Pgr, Agr, Ogr, REgr, Fgr,
DAD, DAS, DAE, DAP,
SNA, SAU, SAH, SNH,
TDA, TUA, TAA, TIT, TPH, TFM,
PIDA, PACC, PADT, PINT, PAVB, PPRV, PDEX, PCON, PEIT, PEPH, PSMN, POTL,
AX, AU, AE, AC, AP, AA,
OIDA, OACC , OADT, OINT, OAVB, OPRV, ODEX, OCON, OEIT, OEPH, OSMN,
REIT, REPH, RENIT, F

-- the above terminal symbols are previously discussed names of domains, groups and
-- generic families

paramDAD, paramDAS, paramDAE, paramDAP,
paramSNA, paramSAU, paramSAH, paramSNH,

-- the above terminal symbols are previously discussed names of parameters
-- expressing assets or subjects

ANYSTRING, ANYNUMBER, <u>Refinement:</u>, _D, _I, <u>Attributes:</u>, <u>Operation:</u>,

-- ANYSTRING means "sensible" text, name or acronym used by the developers to express semantics of the security issues
-- ANYNUMBER means the "right" numeric value
-- for the refinement, attribute or operation sections special underlined keywords are
-- used at the beginning

STS ::= <`Generic`>

P:

<`Generic`> ::= <Prefix >.<BasicGen> | <BasicGen> | <BasicGen>.<Postfix>
<Prefix> ::= <*domain*>. | <*group*>. | <*domain*>.<*group*>.
<Postfix> ::= <*refinement*> | <GenAttribOp> | <*refinement*>.<GenAttribOp>
<BasicGen> ::= <*family*>.<*mnemonic*>.<*description*> |
<*family*>.<*mnemonic*><*derver*>.<*description*> |

```
        <family>.<mnemonic><insnum>.<description> |
        <family>.<mnemonic><derver><insnum>.<description>
<domain>       ::= GNR | CRP | COM | DAB | TTP | SCR | USD
<group>        ::= DAgr | Sgr | Tgr | Pgr | Agr | Ogr | REgr | Fgr
<family>       ::= <DAfamily> | <Sfamily> | <Tfamily> | <Pfamily> | <Afamily> |
                   <Ofamily> | <REfamily> | <Ffamily>
<DAfamily>     ::= DAD | DAS | DAE | DAP
<Sfamily>      ::= SNA | SAU | SAH | SNH
<Tfamily>      ::= TDA | TUA | TAA | TIT | TPH | TFM
<Pfamily>      ::= PIDA | PACC | PADT | PINT | PAVB | PPRV | PDEX |
                   PCON | PEIT | PEPH | PSMN | POTL
<Afamily>      ::= AX | AU | AE | AC | AP | AA
<Ofamily>      ::= OIDA | OACC | OADT | OINT | OAVB | OPRV | ODEX | OCON | OEIT |
                   OEPH | OSMN
<REfamily>     ::= REIT | REPH | RENIT
<Ffamily>      ::= F
<mnemonic>     ::= ANYSTRING | <MnemParameter> |
        <MnemParameter> | <MnemParameter> |
        <MnemParameter><MnemParameter><MnemParameter> |
        <MnemParameter><MnemParameter><MnemParameter><MnemParameter>
        -- the max. number of generic-type parameters is assumed as 4
<MnemParameter>     ::= ANYSTRING[<paramlist>]ANYSTRING
<paramlist> ::= <paramlist> | <genparam>, <paramlist> |
        <paramlist>,<genparam>
        -- list of parameters is recursively defined – generic-type parameter may have many
        -- items assigned
<genparam> ::= <paramFamily> | <paramFamily><=<Generic>
        -- parameter is left uncompleted or has a generic assigned
<paramFamily> ::= paramDAD | paramDAS | paramDAE | paramDAP | paramSNA |
        paramSAU | paramSAH | paramSNH
        -- please note that the allowed parameters represent assets and subjects
<description>::= ANYSTRING
<refinement>  ::= <RefmPrefix>ANYSTRING
<RefmPrefix> ::= Refinement:
<derver> ::= <DerVerPrefix>ANYNUMBER
<DerVerPrefix> ::= "_D"
<insnum> ::= <InsNumPrefix>ANYNUMBER
<InsNumPrefix> ::= "_I"
<GenAttribOp>::= <genattrib> | <genattrib>.<genoper> | <genoper>
<genattrib>::= <AttrPrefix><AttrList>
<genoper>::= <OperPrefix><OperList>
<AttrPrefix>   ::= Attributes:
<OperPrefix>  ::= Operation:
<AttrList>     ::= <AttrItem> | <AttrList>,<AttrItem>
<AttrItem>     ::= <AttrName> | <AttrName><=<AttrValue> |
                   <AttrName>:<AttrType>| <AttrName>:<AttrType><=<AttrValue>
<AttrName>     ::= ANYSTRING
<AttrType>     ::= ANYSTRING
```

```
<AttrValue>   ::= ANYNUMBER | ANYSTRING
<OperList>    ::= <OperItem> | <OperList>,<OperItem>
<OperItem>    ::= <OperName>() | <OperName>():<OperType> |
                  <OperName>(<ArgList>) | <OperName>(<ArgList>):<OperType> |
<OperName>    ::= ANYSTRING
<OperType>    ::= ANYSTRING
<ArgList>     ::= <ArgItem> | <ArgList>,<ArgItem>
<ArgItem>     ::= <ArgName> | <ArgName><=<ArgValue> |
                  <ArgName>: <ArgType>| <ArgName>:<ArgType><=<ArgValue>
<ArgName>     ::= ANYSTRING
<ArgType>     ::= ANYSTRING
<ArgValue>    ::= ANYNUMBER | ANYSTRING.
```

□

The G_G grammar issues the set of possible generics. Let us mark it by Γ. It helps to construct the proper names.

Definition 3.36: Language of generics.

The generics language $\Gamma(G_G)$ specified by the G_G grammar is the set of all generics consisting of the A_T terminal symbols derived from the staring symbol *STS* of the G_G grammar:

$$\Gamma(G_G) = \{\texttt{Generic}\colon \texttt{Generic} \in Z_G \wedge STS \underset{G_G}{\Rightarrow} \texttt{Generic}\},$$

where Z_G is the set of all generics created by the use of the A_T terminal symbols .

□

Please note that the above generics syntax definition is compatible with the earlier discussed generics classes definitions. It is much more precise in comparison with the components definition included within the Common Criteria.

3.2. UML representation of the functional and assurance components

The Common Criteria standard provides developers with a set of components – means for doing security requirements specifications only – functional [39] and assurance [40]. The components are well-defined within the standard in a semiformal way. For this reason only some aspects concerning the components compatibility with generics to integrate them in a common design library will be briefly discussed. For both kinds of components the same and a simplified UML class (abstract) representation is assumed, i.e. `CCSecComponent`.

The UML class `CCSecComponent` $\in$ SICLASS implies the type $t_{\mathrm{CCSecComponent}} \in T_{SIC}$ for which the UML class attributes and operations can be defined in the same way as for the generics.

Definition 3.37: Basic and auxiliary attributes of a CC component.

Let $t \in \mathcal{T}$ be an OCL type, $\mathcal{N}$ is a set of finite, non-empty names over the given alphabet $\mathcal{A}$. The attributes of the class `CCSecComponent` $\in$ $\mathcal{S}$I$\mathcal{C}$LASS, defined as a set $\mathrm{ATT}_{\mathrm{CCSecComponent}}$ of signatures:

$a: t_{\mathrm{CCSecComponent}} \rightarrow t$,

where the attribute name $a \in \mathcal{N}$, and an implied class type $t_{\mathrm{CCSecComponent}} \in \mathcal{T}$, can be expressed as:

$\mathrm{ATT}_{\mathrm{CCSecComponent}} =$

$\{$*class*: $t_{\mathrm{CCSecComponent}} \rightarrow t_{\mathrm{CCclass}}$,

family: $t_{\mathrm{CCSecComponent}} \rightarrow t_{\mathrm{CCFAMILY}}$,

mnemonic: $t_{\mathrm{CCSecComponent}} \rightarrow$ String,

description: $t_{\mathrm{CCSecComponent}} \rightarrow$ String,

refinement: $t_{\mathrm{CCSecComponent}} \rightarrow$ String,

compattrib: $t_{\mathrm{CCSecComponent}} \rightarrow$ String,

compnumber: $t_{\mathrm{CCSecComponent}} \rightarrow$ Integer,

insnum: $t_{\mathrm{CCSecComponent}} \rightarrow$ String,

-- and auxiliary attributes for the component items management (see chapter 3.3)

userdefined: $t_{\mathrm{CCSecComponent}} \rightarrow$ Boolean,

assignstat: $t_{\mathrm{CCSecComponent}} \rightarrow t_{\mathrm{ASSTAT}}\}$, where

$t_{\mathrm{ASSTAT}} \in \mathcal{T}_E$ and the semantics of the t_{ASSTAT} is the function

$I(t_{\mathrm{ASSTAT}}) = \mathrm{literals}(t_{\mathrm{ASSTAT}}) \cup \{\perp\}$.

The following interpretation of literals of the type t_{ASSTAT} is assumed:

$I(e_1)$ = I(NON-EXISTING): "The generic or CC component is not defined yet",

$I(e_2)$ = I(DEFINED): "The generics or CC components exist in the library",

$I(e_3)$ = I(ASSIGNED): "Library element was added to the specification".

□

Definition 3.38: Basic operations of a CC component.

Let $t \in \mathcal{T}$ be an operation result of the OCL type, $t_1, \ldots, t_n \in \mathcal{T}$ be operation parameters of the OCL types, $\mathcal{N}$ is a set of finite, non-empty names over the given alphabet $\mathcal{A}$. The operations of the class `CCSecComponent` $\in$ $\mathcal{S}$I$\mathcal{C}$LASS, defined as a set $\mathrm{OP}_{\mathrm{CCSecComponent}}$ of signatures:

$\omega: t_{\mathrm{CCSecComponent}} \times t_1 \times \ldots \times t_n \rightarrow t$,

where the operation symbol $\omega \in \mathcal{N}$, and $t_{\mathrm{CCSecComponent}} \in \mathcal{T}$ is an implied class type, can be expressed as:

$\mathrm{OP}_{\mathrm{CCSecComponent}} =$

$\{$*compoper*: $t_{\mathrm{CCSecComponent}} \times$ OclType $\times$ OclType$\times \ldots, \times$ OclType $\rightarrow t$,

dispname: $t_{\mathrm{CCSecComponent}} \times$ String $\times$ String $\times$ Integer $\times$ String $\rightarrow$ String$\}$.

□

The first symbol, "*compoper*" represents any operation defined for a CC component, but the second one, the "*dispname*", is used to identify CC components objects.

The semantics of the terms concerned with this *class* depends on the Common Criteria standard version. These terms were described in details in the standard [39-40], and could not be discussed there. For the general overview of the security issues presented in the standard, the enumeration types concerning functional requirements $t_{\mathrm{FunCCclass}}$ (*`FunSecClass`* – Fig.

3.10) and assurance requirements $t_{AssCCclass}$ (`AssSecClass` – Fig. 3.11) will be specified only. Both UML abstract classes are child classes of the `CCSecComponent` that means:

$Set(t_{FunCCclass}) \cup Set(t_{AssCCclass}) = Set(t_{CCclass})$ and $Set(t_{FunCCclass}) \cap Set(t_{AssCCclass}) = \varnothing$.

Definition 3.39: Syntax of the functional CC components classes.

The *class* attribute of the `FunSecClass` $\in$ SICLASS UML class is defined as the enumeration type $t_{FunCCclass} \in T_E$, where:

$literals(t_{FunCCclass})$ = {FAU, FCO, FCS, FDP, FIA, FMT, FPR, FPT, FRU, FTA, FTP}.

□

Definition 3.40: Semantics of the functional CC components classes.

The semantics of the enumeration type $t_{FunCCclass} \in T_E$ is a function

$I(t_{FunCCclass}) = literals(t_{FunCCclass}) \cup \{\perp\}$.

The following interpretation of literals of type $t_{FunCCclass}$ is assumed:

$I(e_1)$ = I(FAU): "Security audit – components dealing with recognizing, recording, storing and analyzing relevant information concerning security activities",

$I(e_2)$ = I(FCO): "Communications – components assuring non-repudiation of both the origin and receipt of the information",

$I(e_3)$ = I(FCS): "Cryptographic support – components expressing key management and cryptographic operation",

$I(e_4)$ = I(FDP): "User data protection – is addressed to access control, information flow control, internal TOE transfer, residual information protection, rollback, stored data integrity, data authentication and import/export from/to the outside TOE security functions",

$I(e_5)$ = I(FIA): "Identification and authentication – encompasses the requirements for functions to establish and verify the claimed user identity",

$I(e_6)$ = I(FMT): "Security management – covers different aspects of the security functions data management (e.g. their attributes, data as banners, etc.)",

$I(e_7)$ = I(FPR): "Privacy – protecting the user against disclosure and misuse of his/her identity by others",

$I(e_8)$ = I(FPT): "Protection of the TOE security functions – covers requirements addressing the integrity and management of the mechanisms that provide these functions, and the integrity of their data",

$I(e_9)$ = I(FRU): "Resource utilization – encompasses the components responsible for resource availability, like processing or storage capability, including fault tolerance, service priority and resource allocation",

$I(e_{10})$ = I(FTA): "TOE access – includes components used to control the user's sessions with the TOE",

$I(e_{11})$ = I(FTP): "Trusted paths or channels – covers requirements addressing trusted communication paths (i.e. between the user and the TOE security function) or trusted communication channels (i.e. between the TOE security function and other trusted IT products)".

□

Definition 3.41: Syntax of the assurance CC components classes.

The *class* attribute of the `AssSecClass` $\in$ $\mathsf{SICLASS}$ UML class is defined as the enumeration type $t_{AssCCclass} \in T_E$, where:

$literals(t_{AssCCclass}) = \{ASE, APE, ACM, ADO, ADV, AGD, ALC, ATE, AVA, AMA\}$.

□

Definition 3.42: Semantics of the assurance CC components classes.

The semantics of the enumeration type $t_{AssCCclass} \in T_E$ is a function

$I(t_{AssCCclass}) = literals(t_{AssCCclass}) \cup \{\perp\}$.

The following interpretation of literals of type $t_{AssCCclass}$ is assumed:

$I(e_1) = I(APE)$: "Contents and coherency of the security target",

$I(e_2) = I(ASE)$: "Contents and coherency of the protection profile",

$I(e_3) = I(ACM)$: "Configuration management – components responsible for the control whether functional requirements and specifications are used in the TOE implementation (integrity of the parts of the TOE, tracking and authorization of changes, etc.)",

$I(e_4) = I(ADO)$: "Delivery and operation – components addressing correct delivery, installation, generation and start-up of the TOE",

$I(e_5) = I(ADV)$: "Development – a very important class concerning the TOE assurance, encompassing components dealing with the possible levels of abstraction (i.e. functional specification, high-level design, implementation representation, low-level design), components for correspondence mapping between various TOE security functions representations, components concerning TOE security policy model, and components of the internal structure of these functions (modularity, layering, minimization of complexity)",

$I(e_6) = I(AGD)$: "Guidance documents – represents the requirements for the TOE user and the TOE administrator guidance documentation",

$I(e_7) = I(ALC)$: "Life cycle support – includes components responsible for the discipline and control of the TOE refinement during its development and maintenance",

$I(e_8) = I(ATE)$: "Tests – represents the requirements concerning test depth and coverage, independent testing by evaluators and functional testing by developers",

$I(e_9) = I(AVA)$: "Vulnerabilities assessment – includes components addressed to cover channels analysis, the possibility of misuse or incorrect configuration, the possibility to defeat probabilistic or permutation mechanisms when applied, and general TOE vulnerability analysis",

$I(e_{10}) = I(AMA)$: "Maintenance of assurance – encompasses the requirements to be applied after the TOE certification, ensuring TOE compliance with its ST when changes occur within the TOE or its environment (e.g. detecting new threats or vulnerabilities, changes in user requirements, and bugs removing from the TOE)".

□

Each Common Criteria class concerns a general group of the security issues. Classes are decomposed (see [39-40]) to families, representing sets of common security issues. Each family contains components, each representing an elementary security requirement that can be used by the developer, though components have elements as well. It was assumed that the reader is familiar with the basic Common Criteria issues.

In the Fig. 3.10 the UML abstract class `FunSecClass` ∈ SICLASS representing the functional component, and the general taxonomy "by the component *class*" is shown. Every CC component *class* has its own families, and these contain their CC components (hundreds of items). The figure shows the four-level model of functional security requirements.

The Fig. 3.10 shows an example of the functional *class* decomposition. The FCS class dealing with the cryptographic support has only two families:

- FSC_COP – Cryptographic operation,
- FCS_CKM – Cryptographic key management

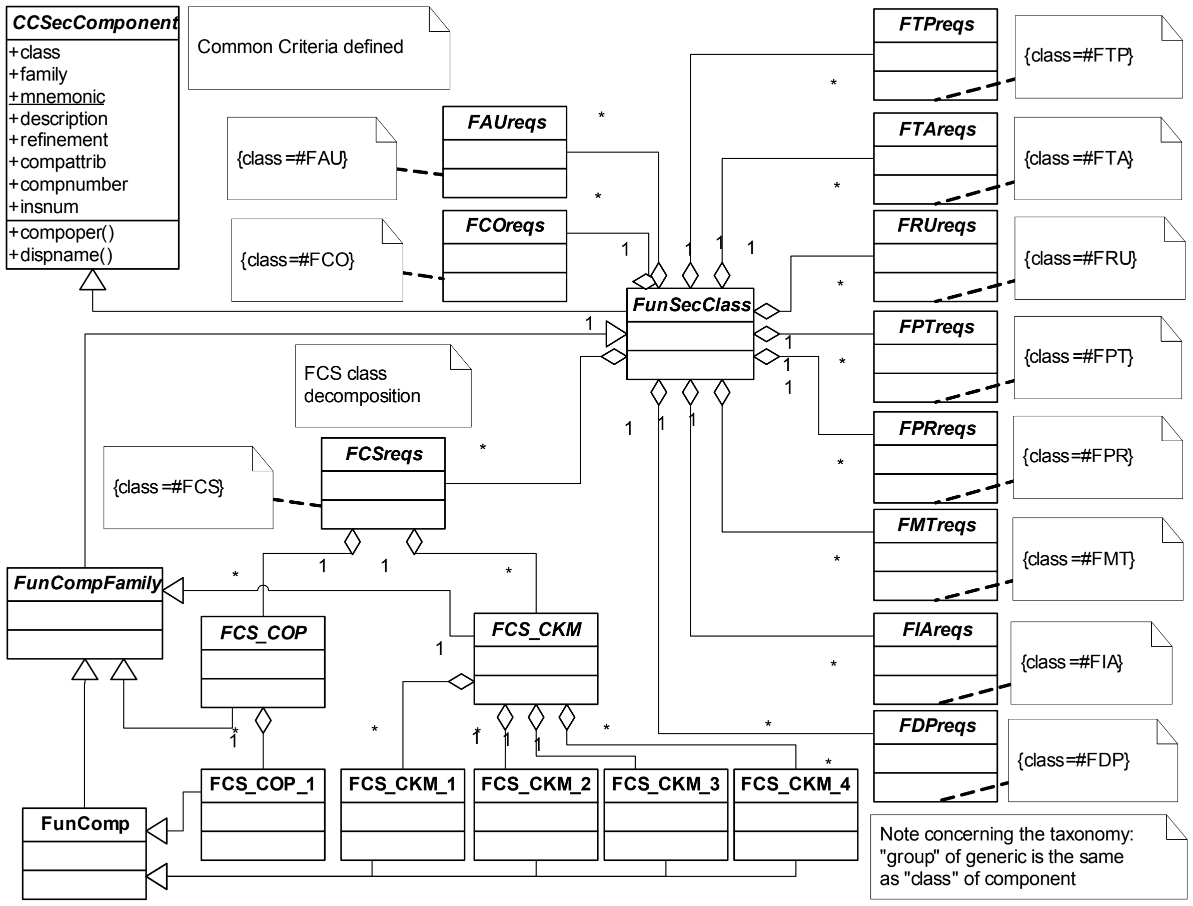


Fig. 3.10. Functional security requirements elements for the TOE and its environment
Rys. 3.10. Wymagania funkcjonalne bezpieczeństwa dla przedmiotu oceny (TOE) i jego środowiska

The first family has only one component having the same name as the family:

- FCS_COP.1[10] – Cryptographic operation.

The second family has four components:

- FCS_CKM.1 – Cryptographic key generation,
- FCS_CKM.2 – Cryptographic key distribution,

[10] To avoid conflicts with OCL dot-separated notation, sometimes the dots in the components name will be replaced by the underscored characters.

- FCS_CKM.3 – Cryptographic key access,
- FCS_CKM.4 – Cryptographic key destruction.

A similar four-level model of security requirements, though dealing with the CC assurance components, is shown in the Fig. 3.11.

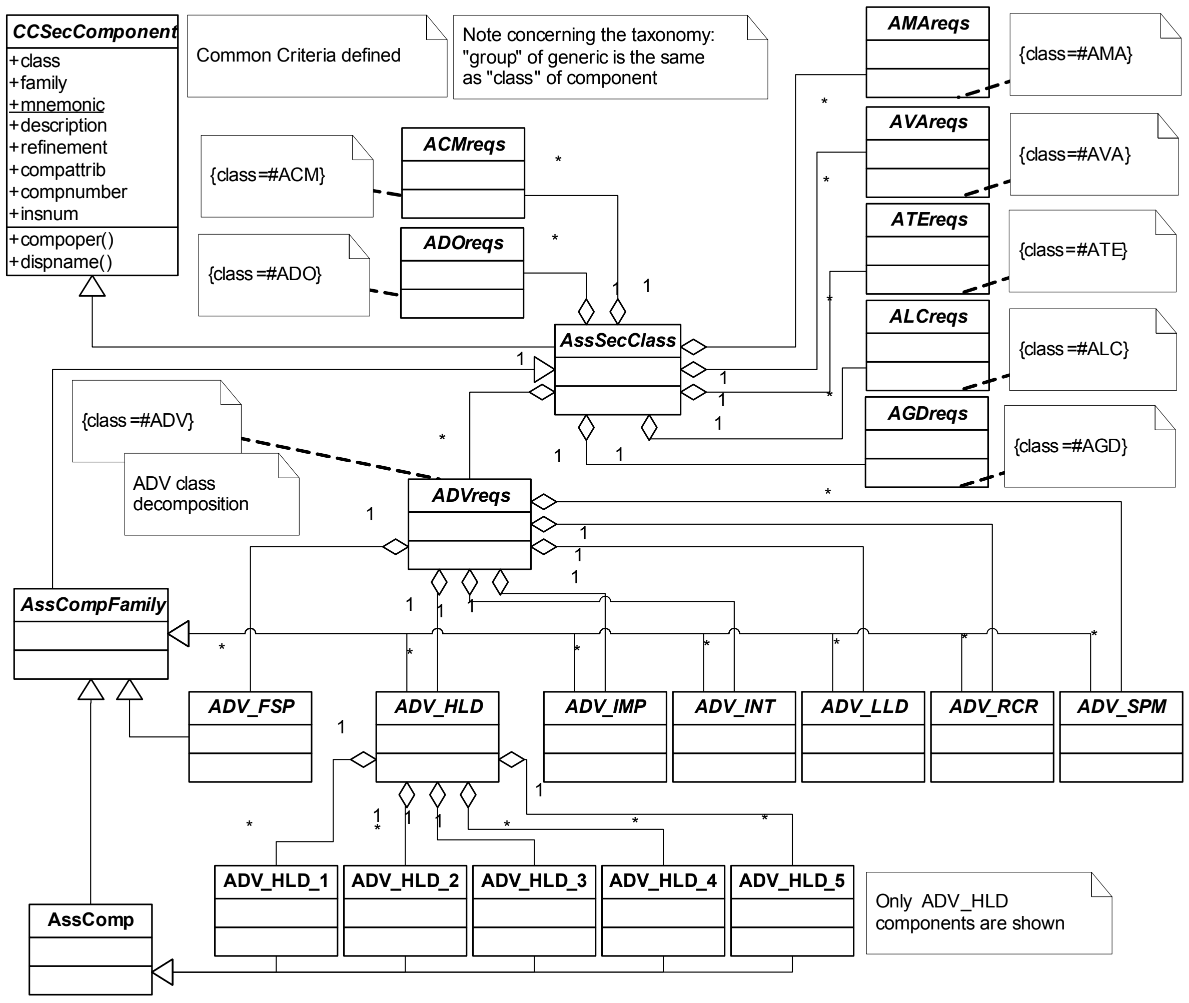


Fig. 3.11. Assurance security requirements elements for the TOE and its environment
Rys. 3.11. Wymagania uzasadniające zaufanie dla przedmiotu oceny (TOE) i jego środowiska

All Assurance components are represented by the abstract class `AssSecClass` ∈ SICLASS. The Fig. 3.11 shows an example of the assurance *class* decomposition.

The ADV *class* dealing with the TOE development has seven families:

- ADV_FSP – Functional specification,
- ADV_HLD – High-level design,
- ADV_IMP – Implementation representation,
- ADV_INT – TSF internals,
- ADV_LLD – Low-level design,
- ADV_RCR – Representation correspondence,
- ADV_SPM – Security policy modelling.

As an example the decomposition of the family ADV_HLD is shown. This family has five components:

- ADV_HLD.1 – Descriptive high-level design,
- ADV_HLD.2 – Security enforcing high-level design,
- ADV_HLD.3 – Semiformal high-level design,
- ADV_HLD.4 – Semiformal high-level explanation,
- ADV_HLD.5 – Formal high-level design.

These figures (Fig. 3.10, Fig. 3.11) deal with the Common Criteria version 2.1. The rearrangement and simplification introduced with the latest 3.1 standard version have no essential meaning for the methodology presented there. The figures present classes of the functional and assurance requirements.

Please note that similar data structures for the generics and components are assumed, which is important for the software implementation. A *class* of Common Criteria components corresponds to a group of generics. Components in the specification, as generics, can be represented by many instances distinguished by the *insnum* attribute. The UML class representing the CC component has an additional operation to issue the developers' style name, i.e. *dispname()*. It can be expressed using the OCL notation, as follows:

```
CCSecComponent
-- assumption: the class attribute family contains
-- the part of CC family name only, located after
-- the prefix "_", e.g."HLD", not "ADV_HLD",
-- for components in the library
self.dispname()=
self.class.concat
    ('_'.concat(family.concat('.'.concat(compnumber))))
-- for components in the specification
self.dispname()=
self.class.concat
    ('_'.concat(family.concat('.'.concat
    (compnumber('_I'.concat(insnum)))))).
□
```

3.3. Generics and components as security specification elements

Please note that the presented above generics and components are only the specification means, a "language" to build real IT security specifications, i.e. the security models (SM). There is a need to distinguish the generics or components placed in the developer's library from those used as the elements of the security model specification, called "assigned". Moreover they can be generics or components of the additional two categories: predefined and placed in the library and defined by developers.

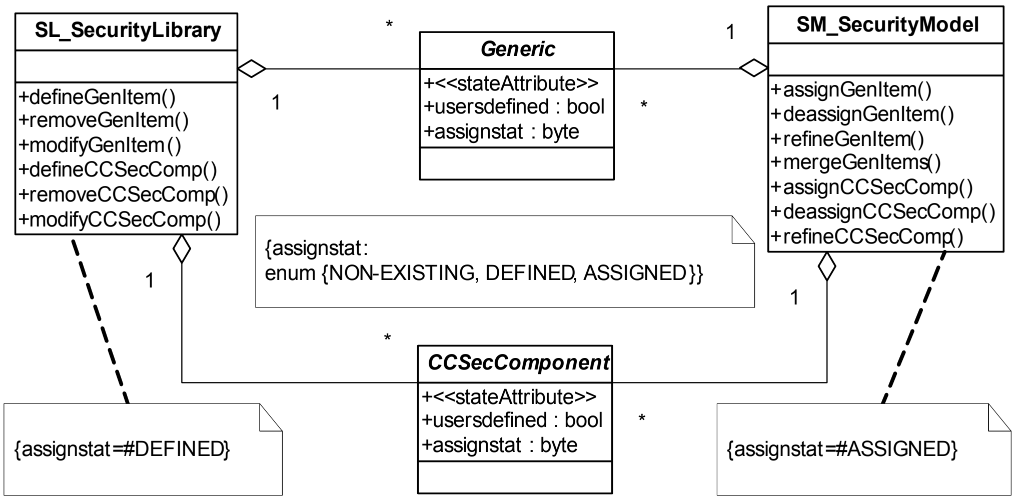


Fig. 3.12. Security items library and security model elements – status flags and operations
Rys. 3.12. Elementy biblioteki bezpieczeństwa i modelu bezpieczeństwa – atrybuty stanu i operacje

To distinguish these cases, special attributes type <<*stateAttribute*>> were added to the `Generic` and `CCSecComponent` classes (Fig. 3.12): *usersdefined*, *assignstat*, and operations to manipulate these flags were added to the `SL_SecurityLibrary` and `SM_SecurityModel` classes. The Boolean *usersdefined* attribute allows to distinguish the generics and/or components defined by the developer from those predefined .

The Fig. 3.13 presents a simplified state machine reflecting the operations results on the generics or components flags. The above flags are used to distinguish the elements defined and placed in the library from those used to build the security specifications.

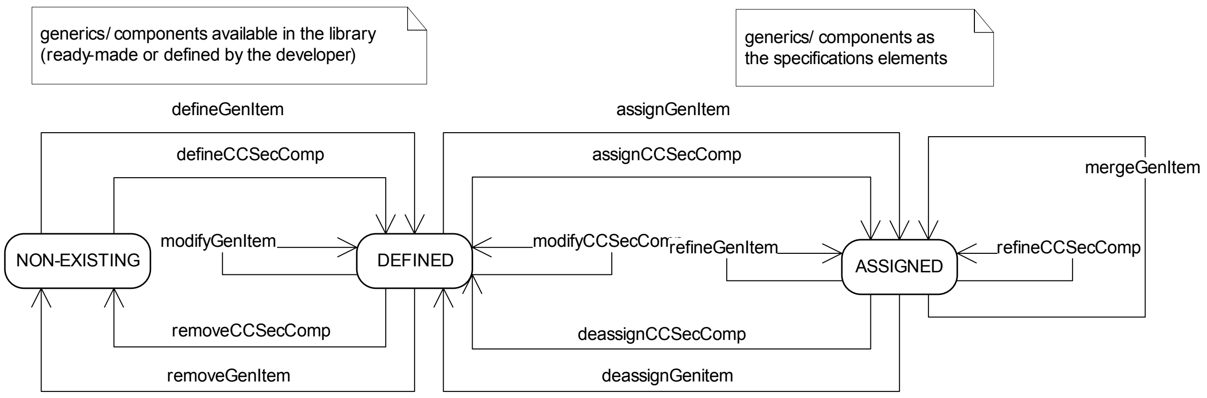


Fig. 3.13. Status of generic or component – main transitions
Rys. 3.13. Stany generyka lub komponentu – przejścia podstawowe

The presented operations: *defineGenItem()*, *removeGenItem()* and *modifyGenItem()*, defined to allow the manipulation on generic-type library elements, can be expressed more precisely using the OCL constraints. This will be shown by a few examples.

```
SL_SecurityLibrary::defineGenItem(c:Generic)
pre: not sL_SecurityLibrary->includes(c) and c.assignstat=#NON-EXISTING
post: sL_SecurityLibrary=
    sL_SecurityLibrary@pre->including(c) and c.assignstat=#DEFINED
□
SL_SecurityLibrary::removeGenItem(c:Generic)
```

```
pre: sL_SecurityLibrary->includes(c) and c.assignstat=#DEFINED
post: sL_SecurityLibrary=
   sL_SecurityLibrary@pre->excluding(c) and c.assignstat=#NON-EXISTING
-- and then the corresponding data structure may be deleted or marked for
-- deletion - depending on the application
□
SL_SecurityLibrary::modifyGenItem(c:Generic)
pre: sL_SecurityLibrary->includes(c) and c.assignstat=#DEFINED
-- required modifications and consistency checkings
post: sL_SecurityLibrary->includes(c) and c.assignstat=#DEFINED
□
```

The operations provided for the manipulation on component-type library elements, i.e. *defineCCSecComp()*, *removeCCSecComp()* and *modifyCCSecComp(),* can be expressed in a very similar way:

```
SL_SecurityLibrary::defineCCSecComp(c:CCSecComponent)
pre: not sL_SecurityLibrary->includes(c) and c.assignstat=#NON-EXISTING
post: sL_SecurityLibrary=
   sL_SecurityLibrary@pre->including(c) and c.assignstat=#DEFINED
□
SL_SecurityLibrary::removeCCSecComp(c:CCSecComponent)
pre: sL_SecurityLibrary->includes(c) and c.assignstat=#DEFINED
post: c sL_SecurityLibrary=
   sL_SecurityLibrary@pre->excluding(c) and c.assignstat=#NON-EXISTING
-- and then the corresponding data structure may be deleted
-- or marked for deletion - depending on the application
□
SL_SecurityLibrary::modifyCCSecComp(c:CCSecComponent)
pre: sL_SecurityLibrary->includes(c) and c.assignstat=#DEFINED
-- required modifications and consistency checkings
post: sL_SecurityLibrary->includes(c) and c.assignstat=#DEFINED
□
```

The defined generic items can be placed in or removed from the security specification. At this moment the item changed its status. These operations, i.e. *assignGenItem()* and *deassignGenItem(),* can be expressed by the following constraints:

```
SM_SecurityModel::assignGenItem(c:Generic)
pre: not SM_SecurityModel->includes(c) and c.assignstat=#DEFINED
post: SM_SecurityModel=
   SM_SecurityModel@pre->including(c) and c.assignstat=#ASSIGNED
□
SM_SecurityModel::deassignGenItem(c:Generic)
pre: SM_SecurityModel->includes(c) and c.assignstat=#ASSIGNED
post: SM_SecurityModel=
   SM_SecurityModel@pre->excluding(c) and
   c.assignstat=#DEFINED
□
```

Two additional operations are proposed for the manipulations on generic-type specification elements: *refineGenItem(), mergeGenItems()*:

```
SM_SecurityModel::refineGenItem(c:Generic)
pre: SM_SecurityModel->includes(c) and c.assignstat=#ASSIGNED
-- required refinements and consistency checkings
post: SM_SecurityModel->includes(c) and c.assignstat=#ASSIGNED
□


SM_SecurityModel::mergeGenItems(main:Generic, aux:Generic)
pre: SM_SecurityModel->includes(main) and main.assignstat=#ASSIGNED and
   SM_SecurityModel->includes(aux) and aux.assignstat=#ASSIGNED
-- when main overlaps aux, both can be merged
-- main item is refined with respect to aux and aux is
-- removed from the specification; consistency checkings of
-- modified main is provided
post: SM_SecurityModel->includes(main) and main.assignstat=#ASSIGNED and
    SM_SecurityModel=
    SM_SecurityModel@pre->excluding(aux) and
    aux.assignstat=#DEFINED
□
```

For the component-type items only three operations to manipulate them are provided, i.e. *assignCCSecComp()*, *deassignCCSecComp()* and *refineCCSecComp()*. They can be expressed using the OCL constraints as well:

```
SM_SecurityModel::assignCCSecComp(c:CCSecComponent)
pre: not SM_SecurityModel->includes(c) and c.assignstat=#DEFINED
post: SM_SecurityModel=
    SM_SecurityModel@pre->including(c) and c.assignstat=#ASSIGNED
□
SM_SecurityModel::deassignCCSecComp(c:CCSecComponent)
pre: SM_SecurityModel->includes(c) and c.assignstat=#ASSIGNED
post: SM_SecurityModel= SM_SecurityModel@pre->excluding(c)
    and c.assignstat=#DEFINED
□
SM_SecurityModel::refineCCSecComp(c:CCSecComponent)
pre: SM_SecurityModel->includes(c) and c.assignstat=#ASSIGNED
-- required refinements and consistency checkings
post: SM_SecurityModel->includes(c) and c.assignstat=#ASSIGNED
□
```

These definitions have very general meaning and deal with all library items or specification items. Please note that the arguments for these operations will always be concrete generics, functional components or assurance components.

3.4. Generics association dealing with generics parameterization

Some groups of generics, i.e. *`TgrGeneric`*, *`PgrGeneric`*, *`AgrGeneric`* and *`OgrGeneric`* have parameters, being *`DAgrGeneric`* or *`SgrGeneric`* generics. The parameterization allows to express different relationships within the security model with respect to the TOE, its environment, internal or external assets, legal or illegal subjects, etc. For example the relationships between threats families and the TOE and its environment were presented in the Fig. 3.14. For every threat a place of undesirable influence is shown, i.e. a place where exploiting of the vulnerabilities may occur.

The Table 3.1 presents the threat scenarios – the relationships between generic families used for the threat specification. Please note threatened assets and threat agents represented by generics. The Table 3.2 presents the relationships between generic families used for the OSPs specification. They express different aspects of right behaviour of the entities (*`SgrGeneric`*) concerning the assets (*`DAgrGeneric`*), i.e. security policy rules.

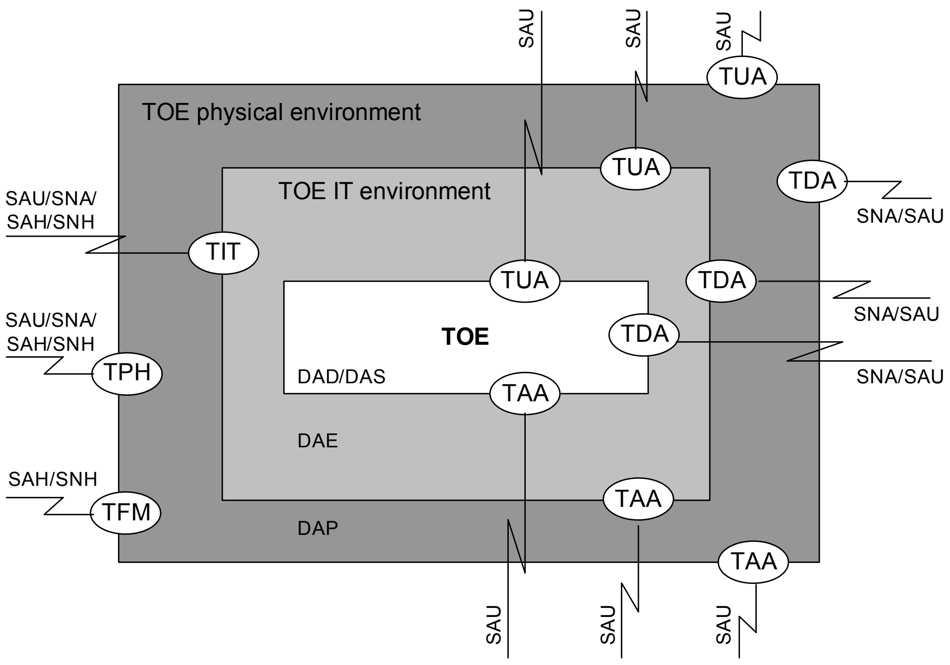


Fig. 3.14. Threat families influencing the TOE and its environment
Rys. 3.14. Rodziny generyków zagrożeń odnoszące się do przedmiotu oceny (TOE) i jego środowiska

The similar *paramDAgr* and *paramSgr* may occur in `AgrGeneric` and `OgrGeneric` generics specification (see proper class attributes in the Fig. 3.2).

Table 3.1
The common relationships between threats, assets and subject families

Threat family	Threatened asset expressed by the parameter (*paramDAgr*)	Subject – the trespasser, expressed by the parameter (*paramSgr*)	Concerns

{*family*=#TUA}	{*family*=#DAD}
	{*family*=#DAS}
		{*family*=#SAU}	Users' errors/mistakes, negligence

{*family*=#TAA}	{*family*=#DAD}
	{*family*=#DAS}
		{*family*=#SAU}	Administrators' errors/mistakes, negligence

{*family*=#TIT}	{*family*=#DAE}	{*family*=#SAU}
		{*family*=#SNA}
		{*family*=#SAH}
		{*family*=#SNH}
			Software (flaws, malicious codes, etc.) and hardware (failures, power disruption, tampering, electromagnetic emanation, etc.) aspects – accidents, failures dealing with the TOE IT environment

{*family*=#TPH}	{*family*=#DAP}	{*family*=#SAU}
		{*family*=#SNA}
		{*family*=#SAH}
		{*family*=#SNH}
			Failures and accidents within the TOE physical environment including the technical infrastructure

{*family*=#TFM}	{*family*=#DAE}	{*family*=#SAH}
		{*family*=#SNH}
			Force majeures, accidents, catastrophes – global-scale undesirable events

{*family*=#TDA}	{*family*=#DAD}
	{*family*=#DAS}
	{*family*=#DAE}
	{*family*=#DAP}
		{*family*=#SNA}
		{*family*=#SAU}
			Direct and intentional attacks on different families of assets (information, availability, IT systems and their physical environment), and also administrators' and users' malicious activities

Please note (see Tables 3.1, 3.2) that not all parameter types (generic family) are allowed for the given threat, OSP, assumption or security objectives. The operation of the value assignment to the parameter is expressed by the symbol "<=". It was assumed that generics (or components) in the library have parameters unassigned. After placing the generics and/or components in the specification (*assignstat*= #ASSIGNED) the parameter can have the right generic assigned or can be left empty. This is similar to the "uncompleted components".

Table 3.2

The common relationships between policies, assets and subject families

OSP family	Ruled asset expressed by the parameter (*paramDAgr*)	Ruling subject expressed by the parameter (*paramSgr*)	Concerns

{*family*=#PIDA}	{*family*=#DAD}
	{*family*=#DAS}
	{*family*=#DAE}
	{*family*=#DAP}
		{*family*=#SAU}
		{*family*=#SNA}
			Identification and authentication of any authorized actor attempting to gain access to the asset

{*family*=#PACC}	{*family*=#DAD}
	{*family*=#DAS}
	{*family*=#DAE}
	{*family*=#DAP}
		{*family*=#SAU}
		{*family*=#SNA}
			Access control and information flow control rules

{*family*=#PADT}	{*family*=#DAD}
	{*family*=#DAS}
	{*family*=#DAE}
	{*family*=#DAP}
		{*family*=#SAU}
		{*family*=#SNA}
			Accountability and security audit

{*family*=#PINT}	{*family*=#DAD}
	{*family*=#DAS}
	{*family*=#DAE}
	{*family*=#DAP}
		{*family*=#SAU}
		{*family*=#SNA}
		{*family*=#SAH}
		{*family*=#SNH}
			Any aspects of integrity (information, services, software, hardware, technical infrastructure)

{*family*=#PAVB}	{*family*=#DAD}
	{*family*=#DAS}
	{*family*=#DAE}
	{*family*=#DAP}
		{*family*=#SAU}
		{*family*=#SNA}
		{*family*=#SAH}
		{*family*=#SNH}
			Any aspects of availability (information, services, software, hardware, technical infrastructure)

{*family*=#PPRV}	{*family*=#DAD}
	{*family*=#DAS}
		{*family*=#SAU}	Authorized actors' privacy while using IT systems and their assets

{*family*=#PDEX}	{*family*=#DAD}	{*family*=#SAU}	Any aspects of the secure data

			exchange by authorized actors
{*family*=#PCON}	{*family*=#DAD}	{*family*=#SAU}	Any aspects of confidentiality of the information owned by authorized actors

{*family*=#PEIT}	{*family*=#DAE}	{*family*=#SAU}
		{*family*=#SNA}
		{*family*=#SAH}
		{*family*=#SNH}
			Right use of software or hardware within the TOE IT environment

{*family*=#PEPH}	{*family*=#DAP}	{*family*=#SAU}
		{*family*=#SNA}
		{*family*=#SAH}
		{*family*=#SNH}
			Right behaviour within the TOE physical environment

{*family*=#PSMN}	{*family*=#DAD}
	{*family*=#DAS}
	{*family*=#DAE}
	{*family*=#DAP}
		{*family*=#SAU}
		{*family*=#SNA}
		{*family*=#SAH}
		{*family*=#SNH}
			Compatibility with the organizational security management systems

{*family*=#POTL}	{*family*=#DAD}
	{*family*=#DAS}
	{*family*=#DAE}
	{*family*=#DAP}
		{*family*=#SAU}
		{*family*=#SNA}
			Legal and technical compatibility with the organization where the TOE will work

The constraints shown in the Table 3.1 or Table 3.2 can be expressed using the OCL language. Assuming that the types of parameter attributes are: *paramDAgr:*`DAgrGeneric` and *paramSgr:*`SgrGeneric`, the constraints can be presented by two examples, each having two variants:

```
TgrGenerics
-- deals with the first row of the Table 3.1/variant #1
if family=#TUA then
    ((paramDAgr.family=#DAD or paramDAgr.family=#DAS or paramDAgr->isEmpty) and
    (paramSgr.family=#SAU or paramSgr->isEmpty))
else
    'Exception: Not allowed combination of generics parameters!'
endif

TgrGenerics
-- deals with the first row of the Table 3.1/variant #2 (using "self")
self.family=#TUA and
    ((paramDAgr.family=#DAD or paramDAgr.family=#DAS or paramDAgr->isEmpty) and
    (paramSgr.family=#SAU or paramSgr->isEmpty))
else
    'Exception: Not allowed combination of generics parameters!'
endif

PgrGenerics
-- deals with the first row of the Table 3.2/variant #1
if family=#PIDA then
    ((paramDAgr.family=#DAD or paramDAgr.family=#DAS or
    paramDAgr.family =#DAE or paramDAgr.family =#DAP or paramDAgr->isEmpty) and
    (paramSgr.family=#SAU or paramSgr.family =#SNA or paramSgr->isEmpty))
else
    'Exception: Not allowed combination of generics parameters!'
endif
PgrGenerics
-- deals with the first row of the Table 3.2/variant #2 (using "self")
```

```
self.family=#PIDA and
   ((paramDAgr.family=#DAD or paramDAgr.family=#DAS or
   paramDAgr.family =#DAE or paramDAgr.family =#DAP or paramDAgr->isEmpty) and
   (paramSgr.family=#SAU or paramSgr.family =#SNA or paramSgr->isEmpty))
else
   'Exception: Not allowed combination of generics parameters!'
Endif
□
```

The parameterization of generics implies special kind of association GenParAssoc, represented by the association classes existing on the generic group level. In the Fig. 3.15 the associations between generics representing their parameterization was shown. Please note the assumed multiplicities and that there were no explicit role names assigned. The multiplicity = 0 can be interpreted as "a generic has a parameter, but the value, i.e. other generic, was not assigned". The class names will be used as the role names.

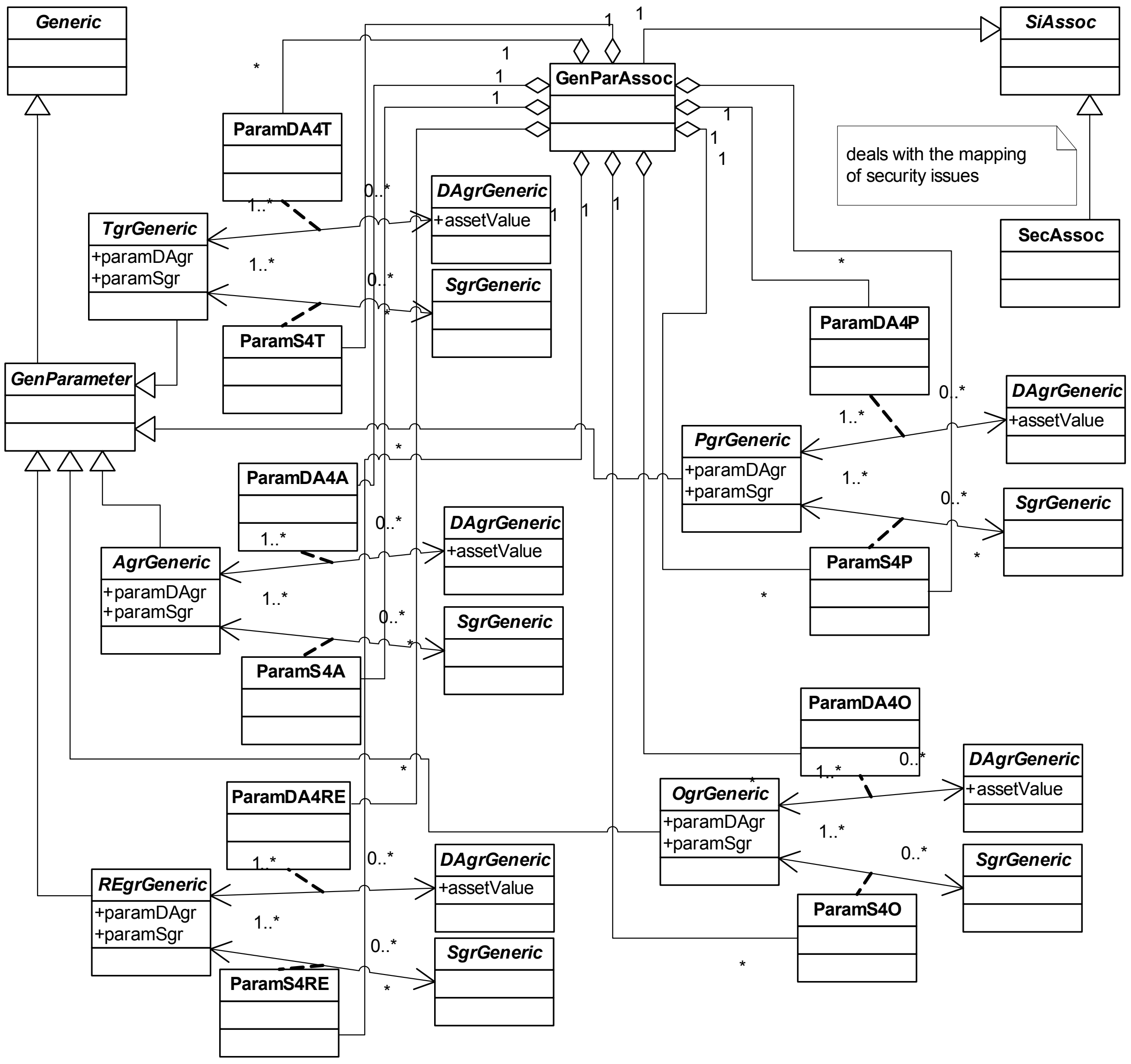


Fig. 3.15. The associations of generics concerning their parameterization
Rys. 3.15. Powiązania generyków dotyczące ich parametryzacji

Example 3.3: Parameterized generics.

TPH.MediaDisposal. Data [paramDAD] is disclosed or inferred from the disposal medium by an unauthorized [paramSNA <= SNA.CleaningPers]
The first *paramDAD* was left unassigned which means “any data classified as DAD family”. The *paramSNA* represents one of the threat agents, for example *SNA.CleaningPers* (Example 3.2) could be optionally assigned (marked by “<=”) in the same way as it was allowed for the components.
□

The parameters presented in the Fig. 3.15 are generic-type parameters. During the IT security development process also other types of parameters are used for components, e.g. numeric, textual, enumerative.

Please note the SiAssoc class representing all kinds of associations and its two subclasses: GenParAssoc expressing generics parameterization and SecAssoc representing mapping of the security issues, discussed later.

3.5. Formal approach to the generics parameterization

Apart from the class issue providing a common description for a set of objects having the same property, there is a need to describe relationships between these classes existing in the library and/or security model. The general formal definition of association is included in the Appendix B. The association is concerned with the $\mathrm{ASSOC} \subseteq \mathcal{N}$, where $\mathcal{N}$ is a set of finite, non-empty names over the given alphabet $\mathcal{A}$ and with the function associates. Assuming full compatibility with the formal OCL model, all possible names ASSOC can be used, though a subset of names $\mathrm{SIASSOC} \subseteq \mathrm{ASSOC}$, representing any associations with the considered security domain is defined.

The first kind of associations, i.e. GenParAssoc is related to the generics parameterization.

Definition 3.43: Associations of generics concerning their parameterization.

The set of association names concerning parameterization of generics $\mathrm{GenParAssoc} \subseteq \mathrm{SIASSOC} \subseteq \mathrm{ASSOC}$ is:

$$\mathrm{GenParAssoc} = \{\mathit{ParamDA4T}, \mathit{ParamS4T}, \mathit{ParamDA4P}, \mathit{ParamS4P}, \mathit{ParamDA4A}, \mathit{ParamS4A}, \mathit{ParamDA4O}, \mathit{ParamS4O}, \mathit{ParamDA4RE}, \mathit{ParamS4RE}\},$$

and the set of related functions associates: $\mathrm{GenParAssoc} \rightarrow \mathrm{SICLASS}^{+}$ can be expressed as:

$$\forall\, as_{ParamDA4T} \in \mathit{ParamDA4T},\ as_{ParamDA4T} \mapsto \langle \mathit{TgrGeneric}, \mathit{DAgrGeneric} \rangle,$$

$$\forall\, as_{ParamS4T} \in \mathit{ParamS4T},\ as_{ParamS4T} \mapsto \langle \mathit{TgrGeneric}, \mathit{SgrGeneric} \rangle,$$

$$\forall\, as_{ParamDA4P} \in \mathit{ParamDA4P},\ as_{ParamDA4P} \mapsto \langle \mathit{PgrGeneric}, \mathit{DAgrGeneric} \rangle,$$

$$\forall\, as_{ParamS4P} \in \mathit{ParamS4P},\ as_{ParamS4P} \mapsto \langle \mathit{PgrGeneric}, \mathit{SgrGeneric} \rangle,$$

$$\forall\, as_{ParamDA4A} \in \mathit{ParamDA4A},\ as_{ParamDA4A} \mapsto \langle \mathit{AgrGeneric}, \mathit{DAgrGeneric} \rangle,$$

$$\forall\, as_{ParamS4A} \in \mathit{ParamS4A},\ as_{ParamS4A} \mapsto \langle \mathit{AgrGeneric}, \mathit{SgrGeneric} \rangle,$$

$\forall$ $as_{ParamDA4O} \in$ *ParamDA4O*, $as_{ParamDA4O} \mapsto \langle$ *OgrGeneric, DAgrGeneric* $\rangle$,
$\forall$ $as_{ParamS4O} \in$ *ParamS4O*, $as_{ParamS4O} \mapsto \langle$ *OgrGeneric, SgrGeneric* $\rangle$,
$\forall$ $as_{ParamDA4RE} \in$ *ParamDA4RE*, $as_{ParamDA4RE} \mapsto \langle$ *REgrGeneric,DAgrGeneric* $\rangle$,
$\forall$ $as_{ParamS4RE} \in$ *ParamS4RE*, $as_{ParamS4RE} \mapsto \langle$ *REgrGeneric,SgrGeneric* $\rangle$.

□

While every associates function encompasses two groups of generic families, called `Parameterized` and `Parameter` classes, all these cases can be expressed in a more generalized way, i.e.

$\forall$ $as_{Param4Parameterized} \in$ GenParAssoc,
$as_{Param4Parameterized} \mapsto \langle$ *Parameterized, Parameter* $\rangle$.

To avoid repeating very similar definitions, the associations concerning parameterization are defined on the generic group level (abstract classes). The GenParAssoc set of associations represents the common issues dealing with the parameterization. These properties are inherited by lower level classes. The particular association deals with the family member classes, where real generic items have links implied by the associations.

The problem is how to define role names and multiplicities. It was assumed that no explicitly defined role names exist and the class names on the generic items level will be used as the role names. According to the UML convention, the class name can be used as a role name, though after changing its first letter to the lower case. Distinction of the role names is assured by the distinction of the generic names.

Definition 3.44: Class name to the role name translation.

Let `ClassName` be a name of the target end class of the association. The role name
roleName(`ClassName`)
can be defined as follows:

```
If ClassName.size=1 then
roleName(ClassName) = ClassName.toLower
else
-- ClassName.size>1
roleName(ClassName) = ((ClassName.substring(1,1)).toLower)).
concat(ClassName.substring(2,ClassName.size))
endif □
```

Definition 3.45: Role names for associations representing parameterization.

Let $as_{Param4Parameterized} \in$ GenParAssoc be an association with the function associates:
$as_{Param4Parameterized} \mapsto \langle$ *Parameterized, Parameter* $\rangle$.
Role names $r_1, r_2, r_3, ..., r_n$ for an association are defined by the function:
roles: GenParAssoc $\rightarrow \mathcal{N}^+$, and $as_{Param4Parameterized} \mapsto \langle r_1, r_2, r_3, ...r_n \rangle$ with $n \geq 2$,
where:
i. all role names ought to be distinct: $\forall i, j \in \{1, ..., n\}: i \neq j \Rightarrow r_i \neq r_j$,
ii. for i=1,
$r_1 =$ `roleName(ClassName)` $\wedge$ `ClassName` $\in$ Set(*Parameterized*),

iii. ∀ i∈{2, ..., n}

r_i= `roleName(ClassName)` ∧ `ClassName` ∈ Set(*`Parameter`*).

□

The function roles($as_{Param4Parameterized}$) = ⟨r_1, r_2, r_3, ...r_n⟩ assigns a unique role name to each class which represents a generic of the right group (see Definition 3.43) participating in the association. The unique role name is the translation result of the considered class name.

Definition 3.46: Multiplicities for associations representing parameterization.

Let $as_{Param4Parameterized}$ ∈ GenParAssoc be an association with the function associates:

$as_{Param4Parameterized}$↦ ⟨*`Parameterized, Parameter`*⟩.

The function multiplicities($as_{Param4Parameterized}$) = ⟨M_1, M_2, M_3, ...M_n⟩ assigns to each class c_i participating in the association a non-empty set M_i⊆N (N represents natural numbers), where:

i. for i=1 M_1 =1; c_1 ∈ Set(*`Parameterized`*),

ii. ∀ i∈{2, ..., n} M_i⊆N; c_i ∈ Set(*`Parameter`*).

□

For the given *`Parameterized`* many generics representing families of the *`Parameter`* can be associated that can be expressed as:

```
Parameterized
self.Parameter = Bag(Parameter).
```
□

The parameterization may concern the navigation over library and/or specification items. While the security models are elaborated, the developers ought to be provided with two kinds of navigation facilities. The first one concerned with parameterization, discussed there, and the second one concerned with mapping the given security item by others, which will be discussed later.

The OCL formal model introduces (see Appendix B) the functions participating and navends. The former one gives the set of associations to which a class belongs. The latter one returns the set of role names that are reachable (navigable) from a class along all associations the class participates in. Both functions can be used directly, but their interpretation concerning parameterization or mapping requires a short discussion.

Please note that the following sets are considered with respect to parameterization:

- SICLASS ⊆CLASS,
- GenParAssoc ⊆ SIASSOC ⊆ ASSOC.

Definition 3.47: Function participating with respect to associations representing parameterization.

Let $as_{Param4Parameterized}$ ∈ GenParAssoc be an association with the function associates:

$as_{Param4Parameterized}$↦ ⟨*`Parameterized, Parameter`*⟩.

Let c be a class $c \in \mathrm{Set}(\mathtt{Parameterized}) \vee c \in \mathrm{Set}(\mathtt{Parameter})$,
and both $\mathrm{Set}(\mathtt{Parameterized}), \mathrm{Set}(\mathtt{Parameter}) \subseteq \mathrm{SICLASS}$.
The function participating(c) can be considered as a function:

$\mathrm{SICLASS} \rightarrow \mathrm{P}(\mathrm{GenParAssoc})$,
$c \mapsto \{as_{\mathrm{Param4Parameterized}} \mid as_{\mathrm{Param4Parameterized}} \in \mathrm{GenParAssoc} \wedge$
$\mathrm{associates}(as_{\mathrm{Param4Parameterized}}) = \langle c_1, \dots, c_n \rangle$
$\exists\, i \in \{1, \dots, n\}: c_i = c\}$.

□

The function participating($\mathtt{Parameterized}$) returns the set of associations the class $\mathtt{Parameterized}$ participates in:

$$\begin{aligned}
\mathrm{participating}(\mathit{TgrGeneric}) = &\ \{as_{\mathrm{ParamDA4T}} \mid as_{\mathrm{ParamDA4T}} \in \mathit{ParamDA4T}\} \\
\cup &\ \{as_{\mathrm{ParamS4T}} \mid as_{\mathrm{ParamS4T}} \in \mathit{ParamS4T}\}, \\
\mathrm{participating}(\mathit{PgrGeneric}) = &\ \{as_{\mathrm{ParamDA4P}} \mid as_{\mathrm{ParamDA4P}} \in \mathit{ParamDA4P}\} \\
\cup &\ \{as_{\mathrm{ParamS4P}} \mid as_{\mathrm{ParamS4P}} \in \mathit{ParamS4P}\}, \\
\mathrm{participating}(\mathit{AgrGeneric}) = &\ \{as_{\mathrm{ParamDA4A}} \mid as_{\mathrm{ParamDA4A}} \in \mathit{ParamDA4A}\} \\
\cup &\ \{as_{\mathrm{ParamS4A}} \mid as_{\mathrm{ParamS4A}} \in \mathit{ParamS4A}\}, \\
\mathrm{participating}(\mathit{OgrGeneric}) = &\ \{as_{\mathrm{ParamDA4O}} \mid as_{\mathrm{ParamDA4O}} \in \mathit{ParamDA4O}\} \\
\cup &\ \{as_{\mathrm{ParamS4O}} \mid as_{\mathrm{ParamS4O}} \in \mathit{ParamS4O}\}, \\
\mathrm{participating}(\mathit{REgrGeneric}) = &\ \{as_{\mathrm{ParamDA4RE}} \mid as_{\mathrm{ParamDA4RE}} \in \mathit{ParamDA4RE}\} \\
\cup &\ \{as_{\mathrm{ParamS4RE}} \mid as_{\mathrm{ParamS4RE}} \in \mathit{ParamS4RE}\}.
\end{aligned}$$

The function participating($\mathtt{Parameter}$) returns the set of associations the class $\mathtt{Parameter}$ participates in:

$\mathrm{participating}(\mathit{DAgrGeneric}) = \{as_{\mathrm{ParamDA4T}} \mid as_{\mathrm{ParamDA4T}} \in \mathit{ParamDA4T}\} \cup$
$\{as_{\mathrm{ParamDA4P}} \mid as_{\mathrm{ParamDA4P}} \in \mathit{ParamDA4P}\} \cup \{as_{\mathrm{ParamDA4A}} \mid as_{\mathrm{ParamDA4A}} \in \mathit{ParamDA4A}\} \cup$
$\{as_{\mathrm{ParamDA4O}} \mid as_{\mathrm{ParamDA4O}} \in \mathit{ParamDA4O}\} \cup \{as_{\mathrm{ParamDA4RE}} \mid as_{\mathrm{ParamDA4RE}} \in \mathit{ParamDA4RE}\}$,

$\mathrm{participating}(\mathit{SgrGeneric}) = \{as_{\mathrm{ParamS4T}} \mid as_{\mathrm{ParamS4T}} \in \mathit{ParamS4T}\} \cup$
$\{as_{\mathrm{ParamS4P}} \mid as_{\mathrm{ParamS4P}} \in \mathit{ParamS4P}\} \cup \{as_{\mathrm{ParamS4A}} \mid as_{\mathrm{ParamS4A}} \in \mathit{ParamS4A}\} \cup$
$\{as_{\mathrm{ParamS4O}} \mid as_{\mathrm{ParamS4O}} \in \mathit{ParamS4O}\} \cup \{as_{\mathrm{ParamS4RE}} \mid as_{\mathrm{ParamS4RE}} \in \mathit{ParamS4RE}\}$.

Please note that the sets of associations are the same when viewed from both sides:
$\mathrm{Set}(\mathrm{participating}(\mathtt{Parameterized})) = \mathrm{Set}(\mathrm{participating}(\mathtt{Parameter}))$.

Definition 3.48: Function navends with respect to the given association representing parameterization.

Let $as_{\mathrm{Param4Parameterized}} \in \mathrm{GenParAssoc}$ be an association with the function associates:
$as_{\mathrm{Param4Parameterized}} \mapsto \langle \mathit{Parameterized}, \mathit{Parameter} \rangle$.
Let c be a class $c \in \mathrm{Set}(\mathtt{Parameterized}) \vee c \in \mathrm{Set}(\mathtt{Parameter})$,
and both $\mathrm{Set}(\mathtt{Parameterized}), \mathrm{Set}(\mathtt{Parameter}) \subseteq \mathrm{SICLASS}$.

The function navends, returning the set of all role names reachable or navigable from a class c over a given association $as_{Param4Parameterized}$, can be defined as:

$$\mathrm{SICLASS} \times \mathrm{GenParAssoc} \rightarrow P(N),$$
$$(c, as_{Param4Parameterized}) \mapsto \{r \mid associates(as_{Param4Parameterized}) = \langle c_1, \dots, c_n \rangle \wedge$$
$$roles(as_{Param4Parameterized}) = \langle r_1, \dots, r_n \rangle \wedge$$
$$\exists i,j \in \{1, \dots, n\}: (i \neq j \wedge c_i = c \wedge r_j = r)\}.$$

□

Please note that role names are derived from the target end class names. The above definition of navends is focused on the role names derived from a pair (class, association) that can be interpreted as a single parameter issue. The generics can have many parameters and models contain many parameterized generics. For this reason, a more comprehensive version of the function navends(c) is introduced that returns the set of all role names reachable from a class c along all associations the class participates in.

Definition 3.49: Function navends with respect to all associations representing parameterization.

Let $as_{Param4Parameterized} \in \mathrm{GenParAssoc}$ be an association with the function associates:

$$as_{Param4Parameterized} \mapsto \langle \mathtt{Parameterized},\ \mathtt{Parameter} \rangle.$$

Let c be a class $c \in Set(\mathtt{Parameterized}) \vee c \in Set(\mathtt{Parameter})$,
and both $Set(\mathtt{Parameterized})$, $Set(\mathtt{Parameter}) \subseteq \mathrm{SICLASS}$.
The function navends(c), returning the set of all role names reachable from a class c along all associations $as_{Param4Parameterized}$, can be defined as:

$$\mathrm{SICLASS} \rightarrow P(N),$$
$$c \mapsto \bigcup_{asParamParam4Parameterized \in participating(c)} navends(c, as_{Param4Parameterized}).$$

□

3.6. Iteration and refinement of generics

The parameterization of generics is the foundation of their iteration. The iteration is the use of the same generic (e.g. threat) in the specification many times (instances are numbered) with different parameters assigned (e.g. assets) to express different aspects of a given security issue. The iteration allows more consistency and has the same possibility as the iteration for the CC components. For any generic the developer can add some comments or details as the generic refinement, to better express the given security issue.

Example 3.4: Iteration and refinement of a generic.

TDA.CrpAnal. Card attacker [paramSNA] may compromise [paramDAD] – user data being encrypted by the TOE or the key needed to calculate the plain text from cipher text.
Refinement: To perform this attack the intruder has to know the cipher text but is neither able

to use the decryption function of the TOE nor to observe the behaviour of the TOE during the cryptographic operation.

This (refined) generic has two parameters allowing to assign different but relevant security issues concerning threatened assets (the first of two is also refined) and attackers, as follows:
DAD.PlainText. Plain document to be encrypted.
Refinement: placed in the smartcard register.
DAD.EncKey. Cryptographic keys used as input parameter for encryption or decryption.
SNA.HighPotenIntrud. Intruder having high level skills, enough resources and deep motivation to perform a deliberate attack.

The following instances of iterated generics can be considered (the refinement is omitted for simplification):
TDA.CrpAnal_I0[11]*. Card attacker [paramSNA <=SNA.HighPotenIntrud] may compromise [paramDAD <=DAD.PlainText] – user data being encrypted by the TOE or the key needed to calculate the plain text from cipher text.*
TDA.CrpAnal_I1. Card attacker [paramSNA <=SNA.HighPotenIntrud] may compromise [paramDAD <=DAD.EncKey] – user data being encrypted by the TOE or the key needed to calculate the plain text from cipher text.
□

Please note that the instances are numbered consecutively in the given security project. The developer can leave a parameter unassigned in the same way as "uncompleted" components [39-40]. In this case it means "any of the relevant". The selection operation, allowed for the components, was not introduced up until now, while it can be easily performed by the iteration.

3.7. Security association – creating the developer's supporting chains

The relationships between the generics and/or components of neighbouring IT security development stages, called the security associations, were added to support the whole development process. These relationships are used to propose the basic items to cover the considered item – all items expressed by generics or components. There is a need to distinguish different kinds of these relations. They are expressed by the association class SecAssoc which is an integral part of the security library (SL) or the security model (SM) – Fig. 3.16. This class has enumeration *<<stateAttribute>> assocstat* attribute expressing the life cycle of the security relationships that can be defined as follows:

Definition 3.50: Semantics of the enumeration attribute concerning security association life cycle.

The semantics of the $t_{\mathrm{AssocSTAT}} \in T_E$ is the function

$$I(t_{\mathrm{AssocSTAT}}) = \mathrm{literals}(t_{\mathrm{AssocSTAT}}) \cup \{\perp\}.$$

The following interpretation of literals of the type t_{AssSTAT} is assumed:

[11] When 0, the simplified name can be used *TDA.CrpAnal*

$l(e_1) = l(\text{NON-EXISTING})$: "The association does not exist",

$l(e_2) = l(\text{PROPOSED})$: "The association exists on the library level only",

$l(e_3) = l(\text{MAPPED})$: "The association is used in the security specification, but was not justified completely yet",

$l(e_4) = l(\text{JUSTIFIED})$: "The association is used in the specification and is fully justified – ready for the rationale". □

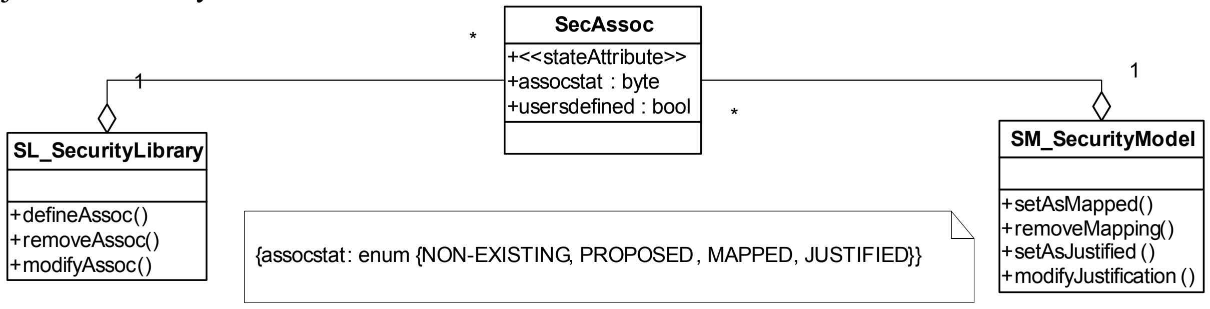


Fig. 3.16. The security association class, its state (enumeration-type) attribute and main operations
Rys. 3.16. Klasa powiązania typu "security association", jej atrybut stanu (typu wyliczeniowego) i podstawowe operacje

The second, Boolean attribute *usersdefined* allows to distinguish the relationships assigned by the developer from the predefined ones. The operations to manipulate associations were added to the `SL_SecurityLibrary` and `SM_SecurityModel` classes. The Fig. 3.17 presents a simplified state machine reflecting the operations results on the relationships between a generic and a generic, or between a generic and a component.

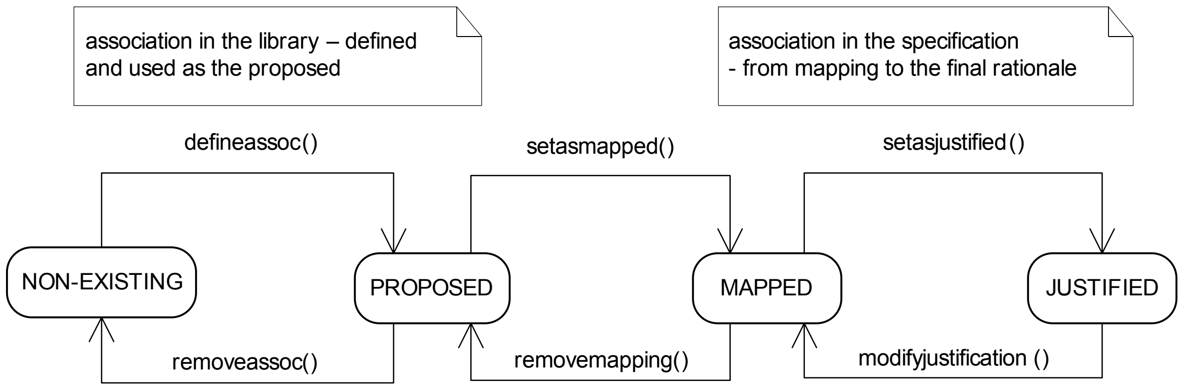

association in the library – defined and used as the proposed
association in the specification - from mapping to the final rationale
defineassoc()
setasmapped()
setasjustified ()
NON-EXISTING
PROPOSED
MAPPED
JUSTIFIED
removeassoc()
removemapping()
modifyjustification ()

Fig. 3.17. Changing the status of the security association – main transitions
Rys. 3.17. Zmiany stanu klasy powiązania typu "security association" – podstawowe przejścia

Different types of relationships, on the generic family level, between generics and/or components imply the set of subclasses of the SecAssoc association class. Please note the assumed multiplicities and that there were no explicit role names assigned. The multiplicity = 0 can be interpreted as "a generic and/or component has a covering issue, but it was intentionally removed from the specification by a developer". The class names will be used as the role names. Please note that every specification element, called principal, can be supported by others, supportive, to fully cover the considered issue. They are usually added by the developer during the rationale process. They will be marked on the class diagram as the UML dependency.

The presented operations: *defineAssoc()*, *removeAssoc()* and *modifyAssoc()*, defined to allow the manipulation on security associations in the library, can be expressed more precisely using the OCL constraints, that will be shown by a few examples.

```
SL_SecurityLibrary::defineAssoc(c:SecAssoc)
pre: not sL_SecurityLibrary->includes(c) and c.assocstat=#NON-EXISTING
post: sL_SecurityLibrary=
   sL_SecurityLibrary@pre->including(c) and c.assocstat=#PROPOSED
□
SL_SecurityLibrary::removeAssoc(c:SecAssoc)
pre: sL_SecurityLibrary->includes(c) and c.assocstat =#PROPOSED`
post: sL_SecurityLibrary=
   sL_SecurityLibrary@pre->excluding(c) and c.assocstat=#NON-EXISTING
-- and then the corresponding data structure may be deleted
-- or marked for deletion – depending on the application
□

SL_SecurityLibrary::modifyAssoc(c:SecAssoc)
pre: sL_SecurityLibrary->includes(c) and c.assocstat=#PROPOSED
-- required modifications and consistency checkings
post: sL_SecurityLibrary->includes(c) and c.assocstat=#PROPOSED
□
```

All "proposed" associations can be used to create "mapped" associations, meaning associations placed into the developed specification. These can be justified and finally "closed" while the entire design is closed after a successful rationale. These operations can also be refined using the OCL constraints:

```
SM_SecurityModel::setAsMapped(c:SecAssoc)
pre: not SM_SecurityModel->includes(c) and c.assocstat=#PROPOSED
post: SM_SecurityModel=
   SM_SecurityModel@pre->including(c) and c.assocstat=#MAPPED
□
SM_SecurityModel::removeMapping(c:SecAssoc)
pre: SM_SecurityModel->includes(c) and c.assocstat=#MAPPED
post: SM_SecurityModel=
   SM_SecurityModel@pre->excluding(c) and c.assocstat=#PROPOSED
□
SM_SecurityModel::setAsJustified(c:SecAssoc)
pre: not SM_SecurityModel->includes(c) and c.assocstat=#MAPPED
post: SM_SecurityModel=
   SM_SecurityModel@pre->including(c) and c.assocstat=#JUSTIFIED
□
SM_SecurityModel::modifyJustification(c:SecAssoc)
pre: SM_SecurityModel->includes(c) and
(c.assocstat=#MAPPED or c.assocstat=#JUSTIFIED)
-- required modification and consistency checkings
post: SM_SecurityModel->includes(c) and c.assocstat=#MAPPED
□
```

The Fig. 3.18 shows default relationships between the security environment and security objectives generics.

The SecAssoc concerning mapping the security issues and GenParAssoc concerning the parameterization are subclasses of abstract SiAssoc class.

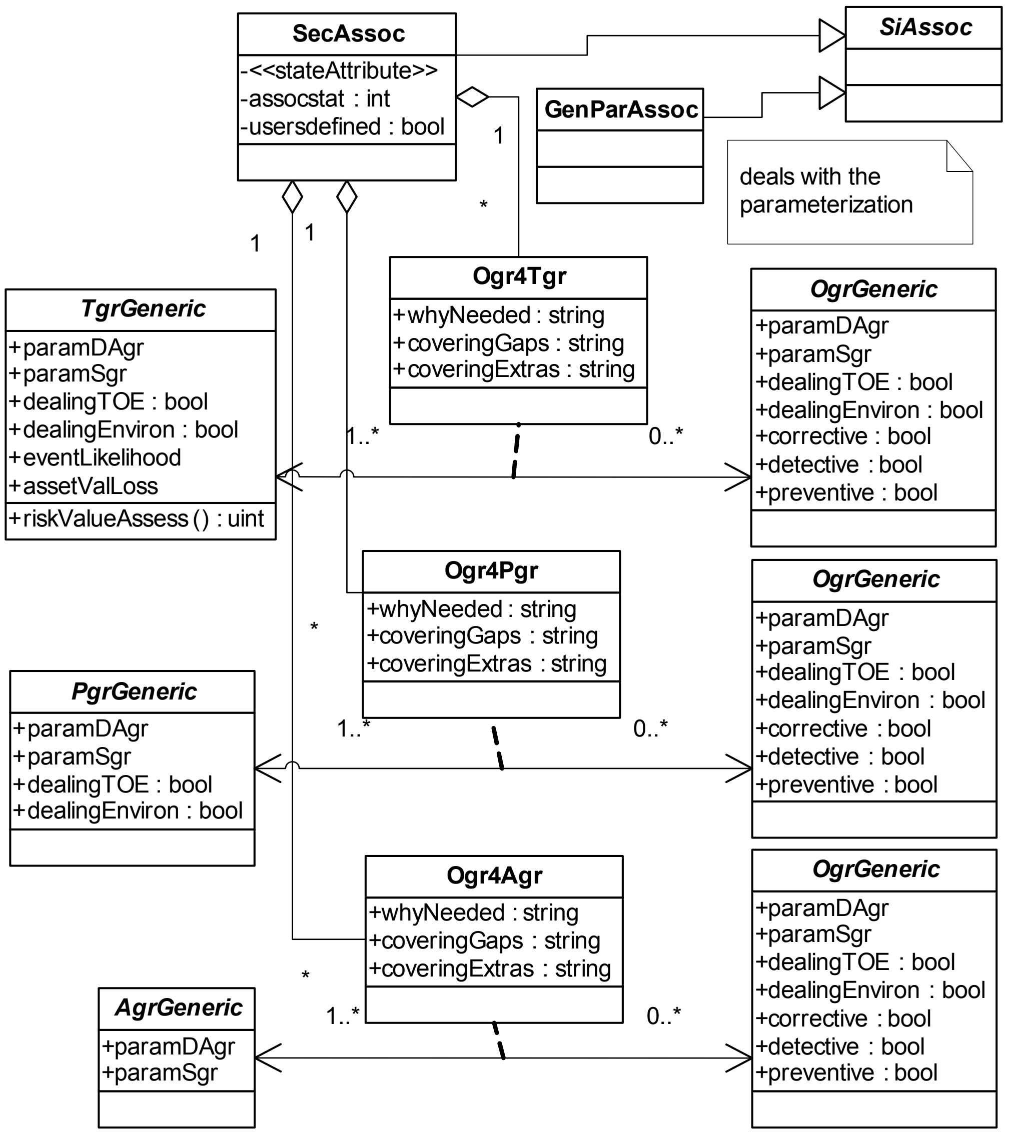


Fig. 3.18. The associations of the security environment and security objectives generics
Rys. 3.18. Powiązania między generykami otoczenia zabezpieczeń i celów zabezpieczeń

The Fig. 3.19 shows the proposed security requirements for the security objectives. Both functional and assurance requirements of the TOE and its environment are considered and expressed by the CC components. Please note `REgrGeneric` elements needed for the TOE environment defined at a very general level.

The Fig. 3.20 shows security functions proposed for the functional security requirements. The functions are strongly dependent on the application character, and for this reason only some of their commonly used examples are defined as generics.

The security associations can be discussed with respect to their placement either in the library or in the model specification. First, let us consider their placement in the security library (SLM).

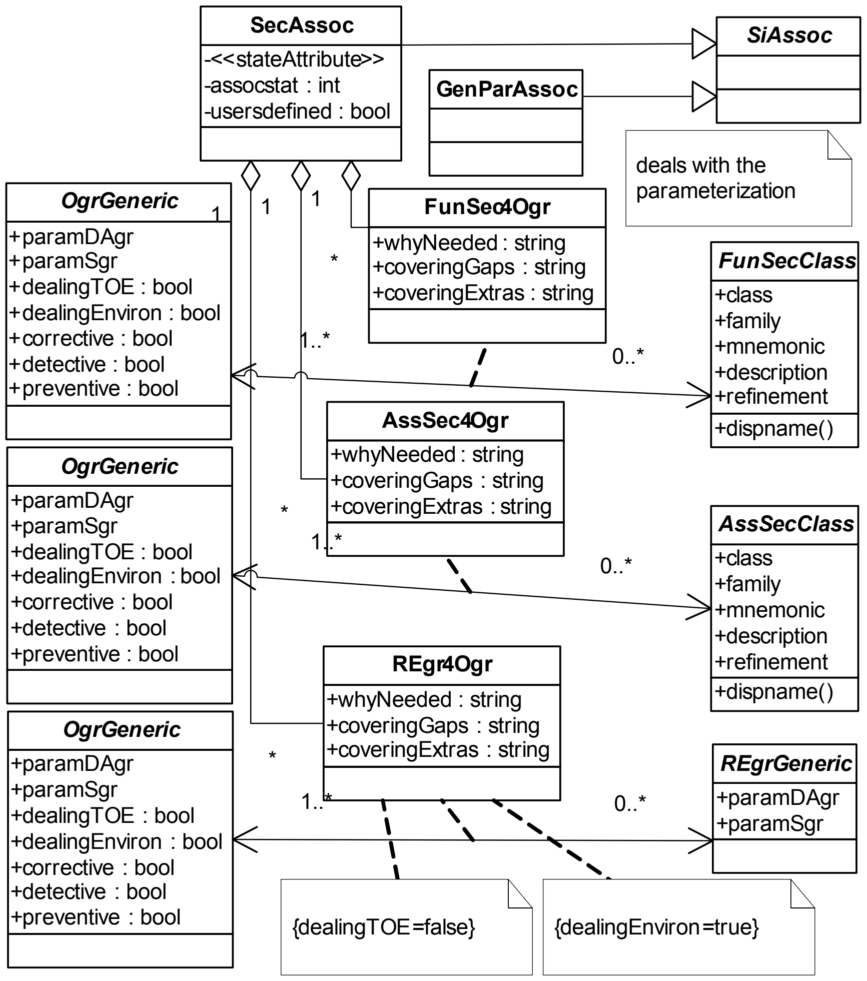


Fig. 3.19. The associations of the security objectives and security requirements
Rys. 3.19. Powiązania między celami zabezpieczeń a wymaganiami bezpieczeństwa

Starting from the security environment generics, through the security objectives and the security requirements, and finally reaching the security functions, the "proposed" generics form chains of elementary security issues. This chain encompasses all proposed designing issues, starting from the problem, ending at the proposed solution. For this reason, such chains will be called "supporting chains". They represent a set of defaults, common issues as a good starting point for any security related design, during which the security specifications are elaborated (PP or ST).

Second, the security associations can be discussed with respect to the security specification (SM model). The proposed generics and/or components, when placed in the security specification, are called "mapped" and have uncompleted justification.

After the final analysis, the justification is completed and they reach the "justified" status. This is expressed by the *assocstat* attribute of the SecAssoc class. The final justification is related to the rationale stage of the IT security development process.

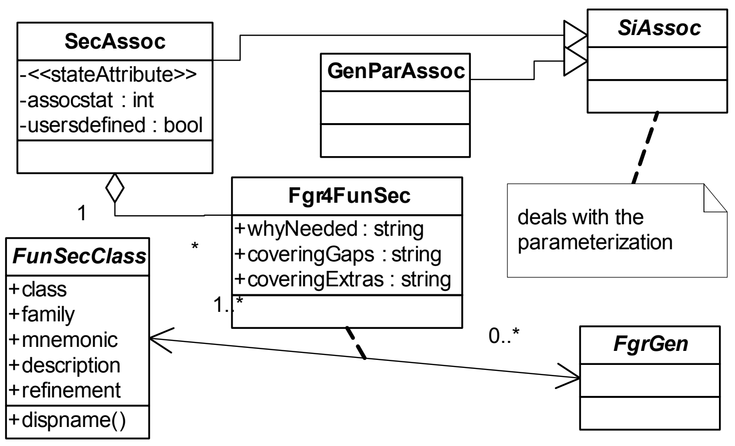


Fig. 3.20. The associations of the security functional requirements and the security functions
Rys. 3.20. Powiązania między wymaganiami funkcjonalnymi bezpieczeństwa i funkcjami zabezpieczającymi

The simple generic chain is presented in the example 3.5.

Example 3.5: The generics relationships chain supporting the IT development process.

The presented chain of the proposed generics and components, covering security items needs, is exemplified by a class diagram shown in the Fig. 3.21, but the Fig. 3.22 presents corresponding generics on an object diagram. Some of the presented definitions are parameterized versions of definitions taken from the real Security Target concerning the smart card controller [87-89].

The entire example, being the continuation of the Example 3.4, concerns one supporting chain started from the `TDAItem` threat family generic item, i.e. *CrpAnal*, though some others are shown too, i.e. generic classes representing parameters or playing supportive roles to the principal ones, and association classes. The Fig. 3.21 shows a part of the simplified security model of a smart card system. This general class diagram, concerning relationships on the group or family level, can be represented on a more detailed level, showing concrete exemplars of generic items and/or CC components as the UML objects (Fig. 3.22), which correspond to the generics in the specification (a security model, please note instances). Please note that the naming convention of the developers' style and the UML style (objects names) is agreed as much as possible. Additionally, please note that the *mnemonic* name with the concatenated derived version and instance number is the unique identifier of the generic of a given family and a group.

Let us consider the chain started at the *TDA.CrpAnal_D0_I0* generic instance (the second instance *TDA.CrpAnal_D0_I1*, not discussed, only shown), representing the real security problem, having two parameters assigned: *DAD.PlainText*, *SNA.HighPotenIntrud.* Other presented security environment generics are:

DAD.EncKey. Cryptographic key stored within the TOE register;
DAD.CipherText. Ciphertext stored within the TOE register;
SAU.CardApp. Smart Card operating system (OS) and its application (AP) residing within the TOE and not being its part.

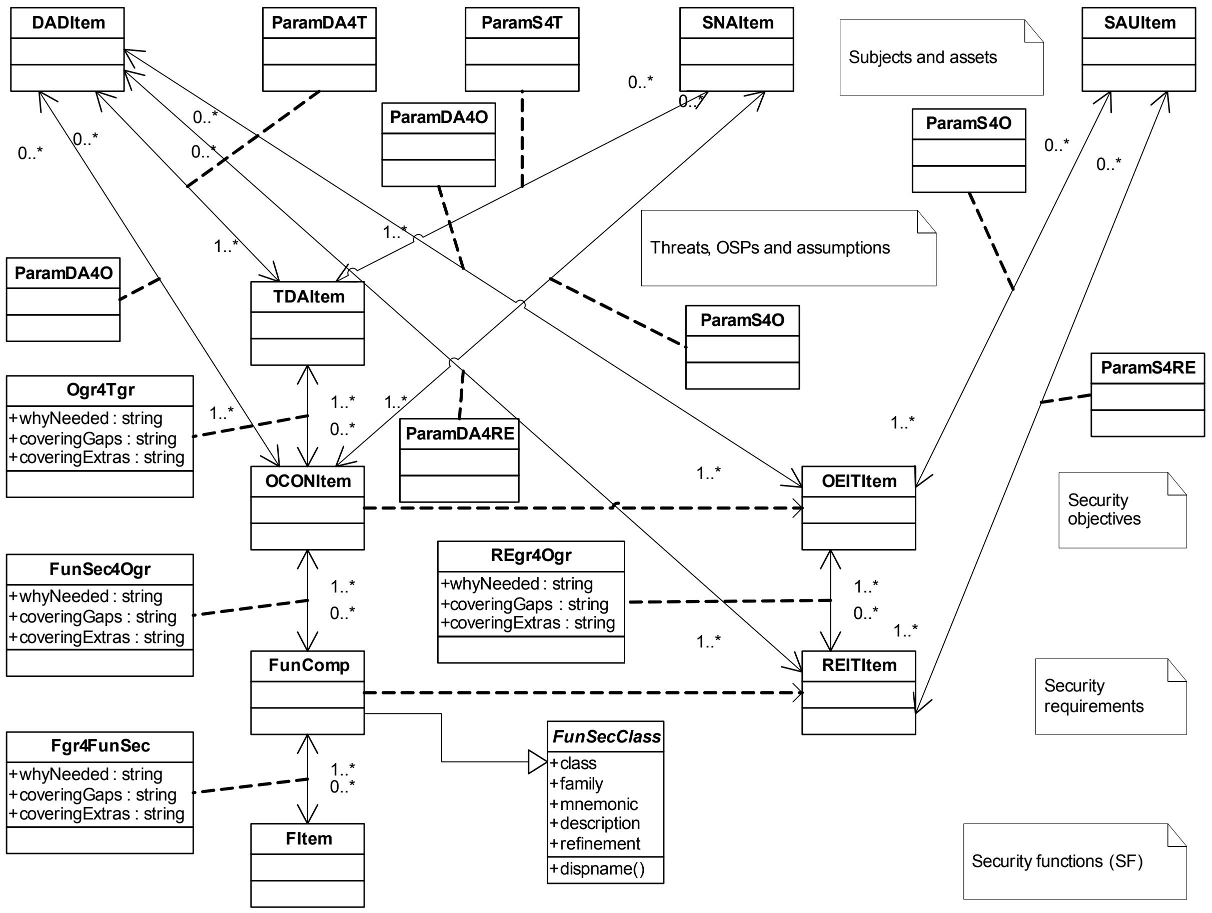


Fig. 3.21. The class diagram concerning the example 3.5 – an example of the generic chain supporting the IT security development process
Rys. 3.21. Diagram klas dotyczący przykładu 3.5 – przykład łańcucha generyków wspomagającego konstruowanie zabezpieczeń

To avoid impacts concerning the *TDA.CrpAnal_D0_I0* threat, the *OCON.BlockCipher* TOE security objective is proposed, supported by the *OEIT.StrongKey, OEIT.RespAppl* and *OEIT.Tamper* TOE IT environment security objectives (see notes on the Fig. 3.22 regarding simplifications), where:

OCON.BlockCipher. The TOE will implement a cryptographic strong symmetric block cipher algorithm to ensure the confidentiality of [paramDAD <= DAD.PlainText] by encryption and to support secure authentication protocols;

OEIT.StrongKey. [paramSAU <= SAU.CardApp] will only use appropriate secret cryptographic keys (chosen from a sufficient key space and with sufficient entropy) as input for the TOE cryptographic function;

OEIT.RespAppl. [paramSAU <= SAU.CardApp] will not disclose security relevant user data, especially the data which will be used as [paramDAD <= DAD.EncKey, DAD.PlainText] to unauthorized users or processes when communicating with a terminal;

OEIT.Tamper. The environment will not expose the TOE to attacks which directly affect or manipulate the device; thus the environment will ensure that security relevant user data and cryptographic keys will not be disclosed and that the random number generator will not be manipulated.

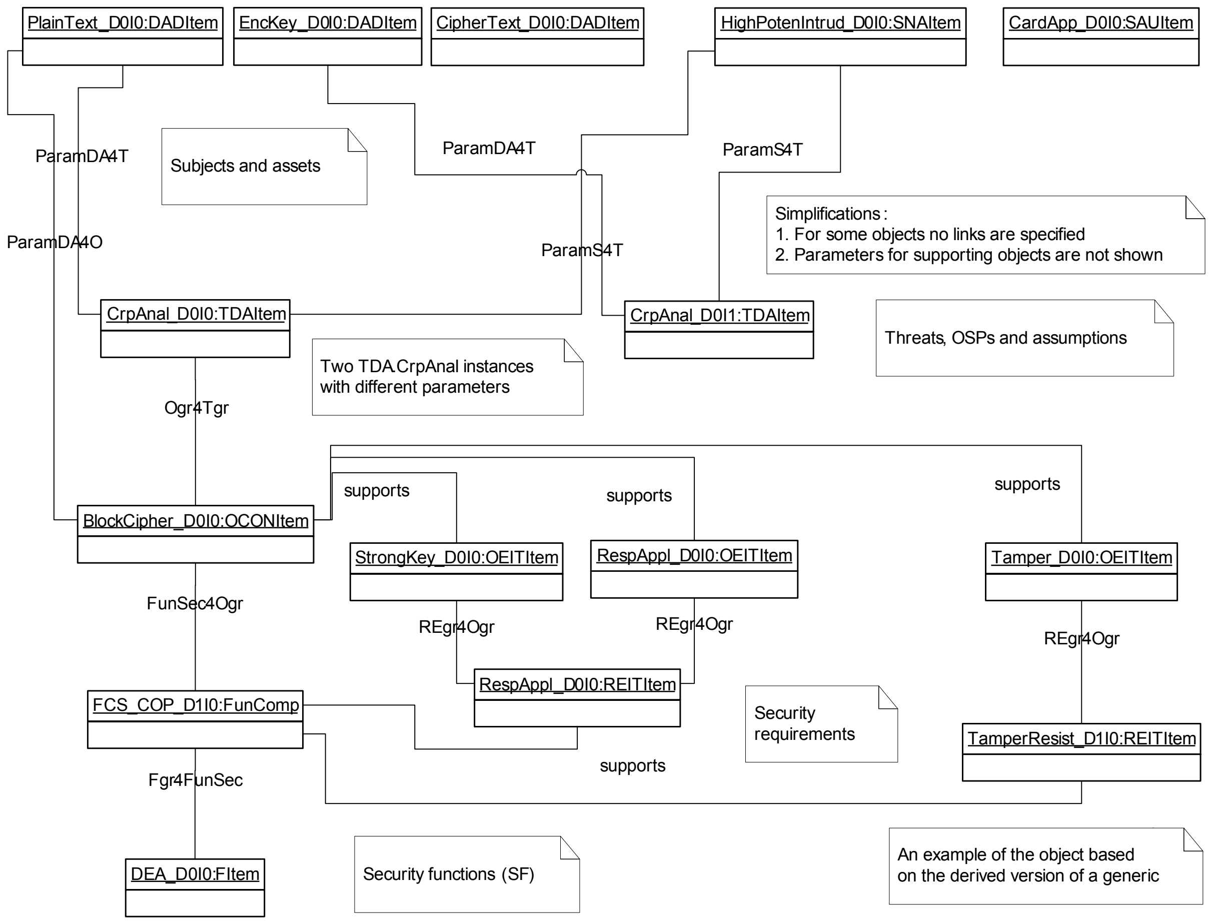


Fig. 3.22. The object diagram concerning the example 3.5 – objects of the generic chain supporting the IT security development process

Rys. 3.22. Diagram obiektów dotyczący przykładu 3.5 – obiekty tworzące łańcuch generyków wspomagający dobór zabezpieczeń

The specified threat is addressed by the four above mentioned objectives, one principal, i.e. *OCON.BlockCipher*, and three others supporting. For the *OCON.BlockCipher* TOE objective the following functional requirement component is proposed with its dependable components *(FDP_ITC.1 or FCS_CKM.1, FCS_CKM.4, FMT_MSA.2* [39] – not discussed/shown there):

> *FCS_COP.1 Cryptographic operation*
>
> *The TSF shall perform [list of cryptographic operations<=encryption and decryption] in accordance with a specified cryptographic algorithm [cryptographic algorithm<= Data Encryption Algorithm (DEA)] and cryptographic key sizes [cryptographic key sizes<=112 bit (Triple DES-Data Encrypting Standard)] that meet the following [list of standards<= FIPS PUB 46, ISO 8732].*

For both *OEIT.StrongKey* and *OEIT.RespAppl* objectives one TOE IT environment requirement is proposed:

> *REIT.RespAppl. Developer of the [paramSAU <=SAU.CardApp] ensures non disclosure of the sensitive data [paramDAD <=DAD.EncKey, DAD.PlainText] to unauthorized subjects [paramSNA[12]] during input/output data operations,*

while for the *OEIT.Tamper* the next TOE IT environment requirement is suggested:

[12] left unrefined; means "any of this type"

REIT.TamperResist_D1[13]*. The TOE environment equipment (i.e. card reader) should provide the TOE physical integrity and enforce its usage in well defined conditions, detecting symptoms of tamper attacks, and ensuring the used equipment (like: card readers) resistance to them.*

It can be said that the specified objectives are addressed by the above mentioned requirements. To meet the functional security requirement *FCS_COP.1* and its dependencies the following security function is proposed:

F.DEA. The DEA is a cryptographic module based on a crypto processor implementing the triple DES algorithm with the key size of 112 bits.

The implementation of the considered supporting chain includes:

TDA.CrpAnal_D0_I0 –> OCON.BlockCipher –> FCS_COP.1 –> F.DEA

generics or components. Other objects, especially concerning the TOE environment, play an auxiliary role.

Please note similarities between generics and components parameterization. The example shows how:

- one of the numerous generics chains, built in the library, can be used to find the common items, covering the given item at any IT security development stage;
- non-parameterized, "flat" generics so far usually used in real STs or PPs, can be parameterized to achieve more flexible and reusable solutions.

□

The above mentioned generics and components deal with the basic (common) relations only, forming the environmental generics – objectives generics – components – and functions generics relationships chain.

During the development process some of the "proposed" solutions can be removed, modified, or quite new can be added to meet specific design needs. The developed library contains hundreds of generics and numerous relationship chains between them. Each of the chains is the solution of an elementary IT security problem.

3.8. Formal approach to the security issues mapping

In the chapter concerning generics parameterization the first kind of association was considered, i.e. $\mathsf{GenParAssoc} \subseteq \mathsf{SIASSOC} \subseteq \mathsf{ASSOC}$. The second group of associations, represented by the $\mathsf{SecAssoc} \subseteq \mathsf{SIASSOC} \subseteq \mathsf{ASSOC}$ and dealing with the mapping of the security issues in the security library and/or security model, will be discussed now.

The general formal definition of association that will be refined there is included in the Appendix B.

Definition 3.51: Associations concerning the mapping of the security issues.

The set of association names dealing with the mapping of the security issues (i.e. generics and/or components) $\mathsf{SecAssoc} \subseteq \mathsf{SIASSOC} \subseteq \mathsf{ASSOC}$ is:

[13] derived generic; both parentheses (..) contain refining

$\mathsf{SecAssoc} = \langle \mathit{Ogr4Tgr}, \mathit{Ogr4Pgr}, \mathit{Ogr4Agr}, \mathit{FunSec4Ogr},$
$\mathit{AssSec4Ogr}, \mathit{REgr4Ogr}, \mathit{Fgr4FunSec} \rangle$,

and the set of related functions associates: $\mathsf{SecAssoc} \rightarrow \mathsf{SICLASS}^+$ can be expressed as:

$\forall\, as_{Ogr4Tgr} \in \mathit{Ogr4Tgr},\ as_{Ogr4Tgr} \mapsto \langle \mathit{TgrGeneric, OgrGeneric} \rangle$,

$\forall\, as_{Ogr4Pgr} \in \mathit{Ogr4Pgr},\ as_{Ogr4Pgr} \mapsto \langle \mathit{PgrGeneric, OgrGeneric} \rangle$,

$\forall\, as_{Ogr4Agr} \in \mathit{Ogr4Agr},\ as_{Ogr4Agr} \mapsto \langle \mathit{AgrGeneric, OgrGeneric} \rangle$,

$\forall\, as_{FunSec4Ogr} \in \mathit{FunSec4Ogr},\ as_{FunSec4Ogr} \mapsto \langle \mathit{OgrGeneric, FunSecClass} \rangle$,

$\forall\, as_{AssSec4Ogr} \in \mathit{AssSec4Ogr},\ as_{AssSec4Ogr} \mapsto \langle \mathit{OgrGeneric, AssSecClass} \rangle$,

$\forall\, as_{REgr4Ogr} \in \mathit{REgr4Ogr},\ as_{REgr4Ogr} \mapsto \langle \mathit{OgrGeneric, REgrGeneric} \rangle$,

$\forall\, as_{Fgr4FunSec} \in \mathit{Fgr4FunSec},\ as_{Fgr4FunSec} \mapsto \langle \mathit{FunSecClass, FgrGeneric} \rangle$.

□

Every associates function encompasses two groups of generics and/or components families, represented by the *IsCovered* and *Covers* classes. The above cases can be generalized and expressed as follows:

$\forall\, as_{Mapping} \in \mathsf{SecAssoc},\ as_{Mapping} \mapsto \langle \mathit{IsCovered, Covers} \rangle$.

The associations concerning mapping are defined at a very general level, i.e. on the generic group level (abstract classes). They represent the common issues dealing with the mapping. These properties are inherited by lower level classes. Real mapping relationships exist on the generic items and components level, where real library and/or specification items have links implied by the associations. This way the repetition of very similar definitions can be avoided.

The role names and multiplicities defining was done in the similar way as for the parameterization. According to the UML convention, it is possible to use the class name as a role name, though its first letter has to be changed to the lower case. Distinction of the role names is assured by the distinction of the generic and component names. The function translating the class name to the role name roleName(`ClassName`) can be used – see Definition 3.44.

Definition 3.52: Role names for associations concerning the mapping of the security issues.

Let $as_{Mapping} \in \mathsf{SecAssoc}$ be an association with the function associates:

$as_{Mapping} \mapsto \langle \mathit{IsCovered, Covers} \rangle$.

Role names $r_1, r_2, r_3, \ldots, r_n$ for an association are defined by the function:

roles: $\mathsf{SecAssoc} \rightarrow \mathcal{N}^+$, and $as_{Mapping} \mapsto \langle r_1, r_2, r_3, \ldots r_n \rangle$ with $n \geq 2$,

where:

i. all role names ought to be distinct: $\forall\, i, j \in \{1, \ldots, n\}: i \neq j \Rightarrow r_i \neq r_j$,

ii. for i=1,

$r_1 = \mathtt{roleName(ClassName)} \wedge \mathtt{ClassName} \in \mathsf{Set}(\mathit{IsCovered})$,

iii. $\forall\, i \in \{2, \ldots, n\}$

$r_i = \mathtt{roleName(ClassName)} \wedge \mathtt{ClassName} \in \mathsf{Set}(\mathit{Covers})$.

□

The function roles($as_{Mapping}$) = ⟨r_1, r_2, r_3, ...r_n⟩ assigns a unique role name to each class which represents a generic of the right group (see Definition 3.51) and/or any functional or assurance component participating in the association.

Definition 3.53: Multiplicities for associations concerning the mapping of the security issues.

Let $as_{Mapping}$ ∈ SecAssoc be an association with the function associates:

$as_{Mapping} \mapsto$ ⟨*IsCovered*, *Covers*⟩.

The function multiplicities($as_{Mapping}$) = ⟨M_1, M_2, M_3, ...M_n⟩ assigns to each class c_i participating in the association a non-empty set $M_i \subseteq N$ (N represents natural numbers), where:

i. for i=1 M_1 =1; c_1 ∈ Set(*IsCovered*),

ii. ∀ i∈{2, ..., n} $M_i \subseteq N$; c_i ∈ Set(*Covers*).

□

For the given `IsCovered` many generics representing families of `Covers` can be associated which can be expressed as:

```
IsCovered
self.Covers = Bag(Covers).
```

Please note that a bag, sometimes called a multi-set, contrary to the set, may contain multiple copies of an element, e.g. {{1,1,2,3,4,4,5,5}}.

During the elaboration of security models, the developers must be provided with two kinds of navigation facilities. The first one is concerned with parameterization and was discussed earlier. The second one is concerned with mapping the given security item by others and will be presented now. The above defined associations concern neighbouring development stages, creating some kind of virtual interfaces between them, especially important for the project rationale. For coverage analysis of the mapped items, the refined functions participating and navends (see Appendix B) can be used.

Definition 3.54: Function participating with respect to associations concerning the mapping of the security issues between two development stages.

Let $as_{Mapping}$ ∈ SecAssoc be an association with the function associates:

$as_{Mapping} \mapsto$ ⟨*IsCovered*, *Covers*⟩.

Let c be a class c∈Set(*IsCovered*) ∨ c∈ Set(*Covers*),
and both Set(*IsCovered*), Set(*Covers*) ⊆ SICLASS.

The function participating(c), can be considered as a function:

SICLASS → P(SecAssoc),

c ↦ {$as_{Mapping}$ | $as_{Mapping}$ ∈ SecAssoc ∧

associates($as_{Mapping}$) = ⟨ c_1, ... , c_n⟩

∃ i ∈ {1, ..., n}: c_i = c}.

□

Let us look at associations from the `IsCovered` class point of view, answering the question “what issue covers the considered one, what items can be used to solve an elementary problem”. The function participating(*IsCovered*) returns the set of associations the class `IsCovered` participates in:

$\text{participating}(\mathit{TgrGeneric}) = \{as_{Ogr4Tgr} \mid as_{Ogr4Tgr} \in \mathit{Ogr4Tgr}\}$,

$\text{participating}(\mathit{PgrGeneric}) = \{as_{Ogr4Pgr} \mid as_{Ogr4Pgr} \in \mathit{Ogr4Pgr}\}$,

$\text{participating}(\mathit{AgrGeneric}) = \{as_{Ogr4Agr} \mid as_{Ogr4Agr} \in \mathit{Ogr4Agr}\}$,

$\text{participating}(\mathit{OgrGeneric}) = \{as_{FunSec4Ogr} \mid as_{FunSec4Ogr} \in \mathit{FunSec4Ogr}\}$
$\cup \{as_{AssSec4Ogr} \mid as_{AssSec4Ogr} \in \mathit{AssSec4Ogr}\}$
$\cup \{as_{REgr4Ogr} \mid as_{REgr4Ogr} \in \mathit{REgr4Ogr}\}$,

$\text{participating}(\mathit{FunSecClass}) = \{as_{Fgr4FunSec} \mid as_{Fgr4FunSec} \in \mathit{Fgr4FunSec}\}$.

Seeing the associations from the `Covers` class viewpoint, the question is “what issue can solve the considered item, why is it needed”. The function participating(*Covers*) returns the set of associations the class `Covers` participates in:

$\text{participating}(\mathit{OgrGeneric}) = \{as_{Ogr4Tgr} \mid as_{Ogr4Tgr} \in \mathit{Ogr4Tgr}\}$
$\cup \{as_{Ogr4Pgr} \mid as_{Ogr4Pgr} \in \mathit{Ogr4Pgr}\}$
$\cup \{as_{Ogr4Agr} \mid as_{Ogr4Agr} \in \mathit{Ogr4Agr}\}$,

$\text{participating}(\mathit{FunSecClass}) = \{as_{FunSec4Ogr} \mid as_{FunSec4Ogr} \in \mathit{FunSec4Ogr}\}$,

$\text{participating}(\mathit{AssSecClass}) = \{as_{AssSec4Ogr} \mid as_{AssSec4Ogr} \in \mathit{AssSec4Ogr}\}$,

$\text{participating}(\mathit{REgrGeneric}) = \{as_{REgr4Ogr} \mid as_{REgr4Ogr} \in \mathit{REgr4Ogr}\}$,

$\text{participating}(\mathit{FgrGeneric}) = \{as_{Fgr4FunSec} \mid as_{Fgr4FunSec} \in \mathit{Fgr4FunSec}\}$.

Definition 3.55: Function navends with respect to the given association concerning the mapping of the security issues between two development stages.

Let $as_{Mapping} \in \text{SecAssoc}$ be an association with the function associates:
$as_{Mapping} \mapsto \langle \mathit{IsCovered}, \mathit{Covers} \rangle$.
Let c be a class $c \in \text{Set}(\mathit{IsCovered}) \vee c \in \text{Set}(\mathit{Covers})$,
and both $\text{Set}(\mathit{IsCovered})$, $\text{Set}(\mathit{Covers}) \subseteq \text{SICLASS}$.
The function navends, returning the set of all role names reachable or navigable from a class c over a given association $as_{Mapping}$, can be defined as:

$\text{SICLASS} \times \text{SecAssoc} \to P(N)$,
$(c, as_{Mapping}) \mapsto \{r \mid \text{associates}(as_{Mapping}) = \langle c_1, \dots, c_n \rangle \wedge$
$\text{roles}(as_{Mapping}) = \langle r_1, \dots, r_n \rangle \wedge$
$\exists i,j \in \{1, \dots, n\}: (i \neq j \wedge c_i = c \wedge r_j = r)\}$.

□

The target end class names are used, in the same way as for the parameterization, to derive the role names. The above definition of navends assumes that the role names derived from a pair (class, association) can be interpreted as a single mapping issue. For more complex relationships with respect to mapping, a more comprehensive version of function $\mathsf{navends}(c)$ is introduced that returns the set of all role names reachable from a class c along all associations the class participates in.

Definition 3.56: Function $\mathsf{navends}$ with respect to all associations concerning the mapping of the security issues between two development stages.

Let $as_{Mapping} \in SecAssoc$ be an association with the function $\mathsf{associates}$:

$as_{Mapping} \mapsto \langle \mathtt{IsCovered}, \mathtt{Covers} \rangle$.

Let c be a class $c \in Set(\mathtt{IsCovered}) \vee c \in Set(\mathtt{Covers})$,
and both $Set(\mathtt{IsCovered}), Set(\mathtt{Covers}) \subseteq \mathrm{SiCLASS}$.
The function $\mathsf{navends}(c)$, returning the set of all role names reachable from a class c along all associations $as_{Mapping}$, can be defined as:

$$\mathrm{SiCLASS} \to P(N),$$
$$c \mapsto \bigcup_{asParamParam4Parameterized \in participating(c)} \mathsf{navends}(c, as_{Mapping}).$$

□

Please note that the navigations are considered there only between classes belonging to the neighbouring development stages.

3.9. Formal approach to the library and security models specification

Summarizing the discussion on internal data representation for the IT Security Development Framework, the full descriptor of classes will be presented. It expresses relationships between hierarchically ordered classes and their properties.

For the security library and the security specifications the four-level taxonomy is assumed (Fig. 3.23):

- high-level description using the abstract items – generics and Common Criteria components – the SiClass encompasses both: `Generic` representing all generics and `CCSecComponent` representing the set of functional and assurance components as a whole specified in the standard; on this level only common properties have been defined;
- groups of generics `GenGroup` and classes of components `FunSecClass`, `AssSecClass` for general classification of these items; please note different meanings of the word "class" in the CC standard and the in the UML modelling domains;
- families of generics `GenFamily` and components `FunCompFamily`, `AssCompFamily` – detailed classification of security issues; associations are considered on this level;
- generic items `GenItem` and functional (`FunComp`)/assurance (`AssComp`) components – concrete items existing in the library or placed in the specifications; when placed into the

security model (specification), they are called "instances" and have attributes *assignstat* = #ASSIGNED.

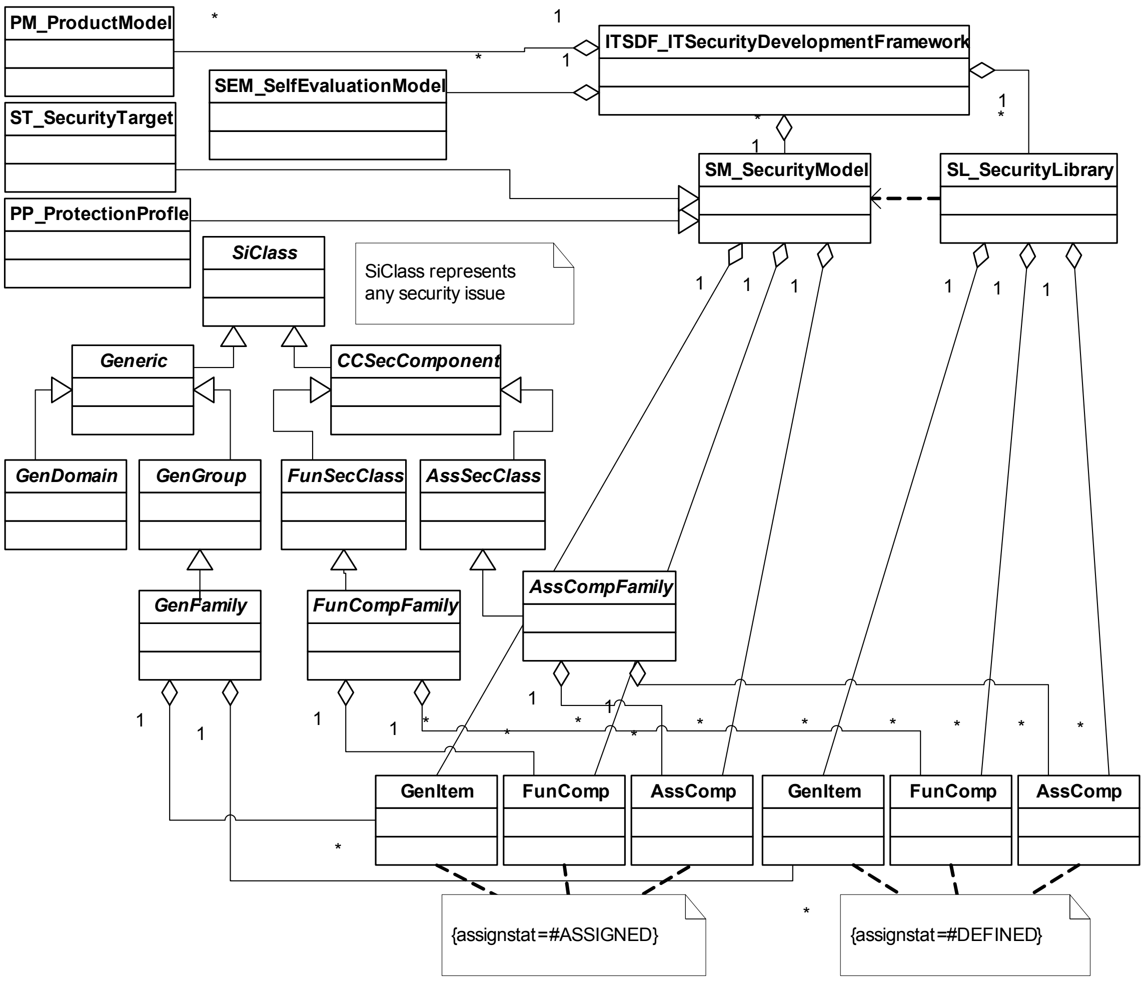


Fig. 3.23. General IT Security Development Framework model – class diagram
Rys. 3.23. Model ogólny szkieletowego systemu konstruowania zabezpieczeń – diagram klas

The first three levels have abstract character (note italicised *names*), allowing to specify common properties. The fourth one expresses the real security items existing in the library and/or specification. The basic four-level hierarchy is created by the UML generalization, which is a taxonomic relationship between two classes (Appendix B). The generalization hierarchy $\prec$ is a partial order on the set of classes $\textsf{SiClass} \subseteq \textsf{Class}$. The child and parent classes are considered. Assuming that classes $c_1, c_2 \in \textsf{Class}$ with $c_1 \prec c_2$; then c_1 is called a child class of c_2, and c_2 is called a parent class of c_1.

Definition 3.57: Function $\mathsf{parents}(c)$ with respect to the considered security domain.

The function $\mathsf{parents}(c)$ is $\textsf{SiClass} \to \mathrm{P}(\textsf{SiClass})$,
$c \mapsto \{c' \mid c' \in \textsf{SiClass} \wedge c \prec c'\}$;
collects all parents of a given class c.
□

With respect to the models of $\mathsf{SiClass}$ discussed there, the following generalization hierarchy can be considered:

$\texttt{GenItem} \prec \textit{GenFamily} \prec \textit{GenGroup} \prec \textit{Generic}$,

$\texttt{FunComp} \prec \textit{FunCompFamily} \prec \textit{FunSecClass} \prec \textit{CCSecComponent}$,

$\texttt{AssComp} \prec \textit{AssCompFamily} \prec \textit{AssSecClass} \prec \textit{CCSecComponent}$.

For the three child classes the full descriptors FD_{GenItem}, FD_{FunComp}, FD_{AssComp} will be specified, which provide detailed specifications of the whole security items encompassed by the $\mathsf{SiClass} \subseteq \mathsf{Class}$.

Definition 3.58: Full descriptor of a class – see Appendix B.

The full descriptor of a class $c \in \mathsf{Class}$ is a structure $FD_c = (\mathrm{Att}^*_c, \mathrm{Op}^*_c, \mathrm{navends}^*(c))$, containing all attributes, operations and navigable role names for a considered class c and all of its parents.

□

With the assumed above general definition of a full descriptor of a class, particular descriptors will be specified.

Definition 3.59: Full descriptor of the class `GenItem`.

The full descriptor of the class `GenItem` $\in \mathsf{SiClass}$ is a structure:

$FD_{\mathrm{GenItem}} = (\mathrm{Att}^*_{\mathrm{GenItem}}, \mathrm{Op}^*_{\mathrm{GenItem}}, \mathrm{navends}^*(\texttt{GenItem}))$, where:

i. attributes: $\mathrm{Att}^*_{\mathrm{GenItem}} = \mathrm{Att}_{\mathrm{GenItem}} \cup \mathrm{Att}_{\mathrm{GenFamily}} \cup \mathrm{Att}_{\mathrm{GenGroup}} \cup \mathrm{Att}_{\mathrm{Generic}}$,
$\mathrm{Att}_{\mathrm{GenItem}} = \varnothing$, $\mathrm{Att}_{\mathrm{GenFamily}}$ – different for the groups – see Definitions 3.11, 3.14, 3.17, 3.20, 3.23, 3.26, 3.29, 3.32, $\mathrm{Att}_{\mathrm{GenGroup}} = \varnothing$, $\mathrm{Att}_{\mathrm{Generic}}$ – see Definition 3.5;

ii. operations: $\mathrm{Op}^*_{\mathrm{GenItem}} = \mathrm{Op}_{\mathrm{GenItem}} \cup \mathrm{Op}_{\mathrm{GenFamily}} \cup \mathrm{Op}_{\mathrm{GenGroup}} \cup \mathrm{Op}_{\mathrm{Generic}}$,
$\mathrm{Op}_{\mathrm{GenItem}} = \varnothing$, $\mathrm{Op}_{\mathrm{GenFamily}}$ – different for the groups – see Definitions 3.11, 3.14, 3.17, 3.20, 3.23, 3.26, 3.29, 3.32, $\mathrm{Op}_{\mathrm{GenGroup}} = \varnothing$, $\mathrm{Op}_{\mathrm{Generic}}$ – see Definition 3.6;

iii. navigable role names: $\mathrm{navends}^*(\texttt{GenItem}) = \mathrm{navends}(\texttt{GenItem}) \cup \mathrm{navends}(\textit{GenFamily}) \cup \mathrm{navends}(\textit{GenGroup}) \cup \mathrm{navends}(\textit{Generic})$,
$\mathrm{navends}(\texttt{GenItem}) = \varnothing$, $\mathrm{navends}(\textit{GenFamily})$ – see Definition 3.56, $\mathrm{navends}(\textit{GenGroup})$ – see Definition 3.49, $\mathrm{navends}(\textit{Generic}) = \varnothing$.

□

Definition 3.60: Full descriptor of the class `FunComp`.

The full descriptor of the class `FunComp` $\in \mathsf{SiClass}$ is a structure:

$FD_{\mathrm{FunComp}} = (\mathrm{Att}^*_{\mathrm{FunComp}}, \mathrm{Op}^*_{\mathrm{FunComp}}, \mathrm{navends}(\texttt{FunComp}))$, where:

i. attributes: $\mathrm{Att}^*_{\mathrm{FunComp}} = \mathrm{Att}_{\mathrm{FunComp}} \cup \mathrm{Att}_{\mathrm{FunCompFamily}} \cup \mathrm{Att}_{\mathrm{FunSecClass}} \cup \mathrm{Att}_{\mathrm{CCSecComponent}}$,
$\mathrm{Att}_{\mathrm{FunComp}} = \varnothing$, $\mathrm{Att}_{\mathrm{FunCompFamily}} = \varnothing$, $\mathrm{Att}_{\mathrm{FunSecClass}} = \varnothing$,
$\mathrm{Att}_{\mathrm{CCSecComponent}}$ – see Definition 3.37;

ii. operations: $\mathrm{Op}^*_{\mathrm{FunComp}} = \mathrm{Op}_{\mathrm{FunComp}} \cup \mathrm{Op}_{\mathrm{FunCompFamily}} \cup \mathrm{Op}_{\mathrm{FunSecClass}} \cup \mathrm{Op}_{\mathrm{CCSecComponent}}$,
$\mathrm{Op}_{\mathrm{FunComp}} = \varnothing$, $\mathrm{Op}_{\mathrm{FunCompFamily}} = \varnothing$, $\mathrm{Op}_{\mathrm{FunSecClass}} = \varnothing$,
$\mathrm{Op}_{\mathrm{CCSecComponent}}$ – see Definition 3.38;

iii. navigable role names: $\mathrm{navends}^{*}$(`FunComp`) = navends(`FunComp`) $\cup$ navends(*`FunCompFamily`*) $\cup$ navends(*`FunSecClass`*) $\cup$ navends(*`CCSecComponent`*), navends(`FunComp`) = $\varnothing$, navends(*`FunCompFamily`*) – can be defined in a similar way as in Definition 3.56, navends(*`FunSecClass`*) – can be defined in a similar way as in Definition 3.49, navends(*`CCSecComponent`*).

□

Definition 3.61: Full descriptor of the class `AssComp`.

The full descriptor of the class `AssComp` $\in$ SICLASS is a structure:

$FD_{AssComp} = (ATT^{*}_{AssComp}, OP^{*}_{AssComp}, \mathrm{navends}^{*}(\texttt{AssComp}))$, where:

i. attributes: $ATT^{*}_{AssComp} = ATT_{AssComp} \cup ATT_{AssCompFamily} \cup ATT_{AssSecClass} \cup ATT_{CCSecComponent}$,

$ATT_{AssComp} = \varnothing$, $ATT_{AssCompFamily} = \varnothing$, $ATT_{AssSecClass} = \varnothing$,

$ATT_{CCSecComponent}$ – see Definition 3.37;

ii. operations: $OP^{*}_{AssComp} = OP_{AssComp} \cup OP_{AssCompFamily} \cup OP_{AssSecClass} \cup OP_{CCSecComponent}$,

$OP_{AssComp} = \varnothing$, $OP_{AssCompFamily} = \varnothing$, $OP_{AssSecClass} = \varnothing$,

$OP_{CCSecComponent}$ – see Definition 3.38;

iii. navigable role names: $\mathrm{navends}^{*}$(`AssComp`) = navends(`AssComp`) $\cup$ navends(*`AssCompFamily`*) $\cup$ navends(*`AssSecClass`*) $\cup$ navends(*`CCSecComponent`*), navends(`AssComp`) = $\varnothing$, navends(*`AssCompFamily`*) – can be defined in a similar way as in Definition 3.56, navends(*`AssSecClass`*) – can be defined in a similar way as in Definition 3.49, navends(*`CCSecComponent`*).

□

Each of the above defined full descriptors $FD_{GenItem}$, $FD_{FunComp}$, $FD_{AssComp}$ ought to satisfy the following conditions:

i. Attributes are defined in exactly one class:

$\forall (a: t_c \rightarrow t,\ a': t_{c'} \rightarrow t' \in ATT^{*}_{c}): (a=a' \Rightarrow t=t_{c'} \wedge t=t')$.

The condition i. is satisfied because attributes sets for particular descriptors are disjointed:

$ATT_{GenItem} \cap ATT_{GenFamily} \cap ATT_{GenGroup} \cap ATT_{Generic} = \varnothing$,

$ATT_{FunComp} \cap ATT_{FunCompFamily} \cap ATT_{FunSecClass} \cap ATT_{CCSecComponent} = \varnothing$,

$ATT_{AssComp} \cap ATT_{AssCompFamily} \cap ATT_{AssSecClass} \cap ATT_{CCSecComponent} = \varnothing$.

ii. An operation may only be defined once:

$\forall (\omega: t_c \times t_1 \times \ldots \times t_n \rightarrow t,\ \omega: t_{c'} \times t_1 \times \ldots \times t_n \rightarrow t' \in OP^{*}_{c}):(t_c=t_{c'})$.

The condition ii. is satisfied because operations sets for particular descriptors are disjointed:

$OP_{GenItem} \cap OP_{GenFamily} \cap OP_{GenGroup} \cap OP_{Generic} = \varnothing$,

$OP_{FunComp} \cap OP_{FunCompFamily} \cap OP_{FunSecClass} \cap OP_{CCSecComponent} = \varnothing$,

$OP_{AssComp} \cap OP_{AssCompFamily} \cap OP_{AssSecClass} \cap OP_{CCSecComponent} = \varnothing$.

iii. Role names are defined in exactly one class:

$\forall\ c_1, c_2 \in \mathrm{parents}(c) \cup \{c\}: (c_1 \neq c_2 \Rightarrow \mathrm{navends}(c_1) \cap \mathrm{navends}(c_2) = \varnothing)$.

The condition iii. is satisfied because role names are derived from class names and these must be unique.

iv. Role names and attribute names must not conflict:

$\forall (a: t_c \rightarrow t \in \mathrm{ATT}^*_c) \wedge \forall r \in \mathrm{navends}^*(c): (a \neq r)$.

For all classes defined there the condition iv is satisfied.

There are hundreds of predefined generics and CC components, and many user-defined ones. For all of them this condition ought to be satisfied by design. It is also a recommendation how to construct names for generics and/or components.

The all previously defined terms allow to define the syntax of the security-related object model $\mathcal{M}_{Si}$ representing any security issue that can be placed in the security library (i.e. concerning the `SL_SecurityLibrary`) and/or in the security specifications (i.e. concerning the SL_SecurityModel). It can be defined analogically to the way the general formal syntax of the object model $\mathcal{M}$ for the OCL language was defined (see Appendix B). The $\mathcal{M}_{Si}$ model is a part of the $\mathcal{M}$ model, and for this reason all OCL data representations and expressions can be used for $\mathcal{M}_{Si}$ as well.

Definition 3.62: Syntax of the security-related object model.

Let $c \in \mathrm{SICLASS} \subseteq \mathrm{CLASS}$ be a class representing a given security-related issue. The syntax of the security-related object models is a structure of the above defined elements:

$\mathcal{M}_{Si} = (\mathrm{SICLASS},\ \mathrm{ATT}_c,\ \mathrm{OP}_c,\ \mathrm{SIASSOC},\ \mathrm{associates},\ \mathrm{roles},\ \mathrm{multiplicities},\ \prec)$.

□

In the UML/OCL the class $c \in \mathrm{CLASS}$ represents the set of objects sharing the same properties. This set is called the domain of the class c.

The objects ought to have unique identifiers assigned. The class c is represented by the infinite set of its objects identifiers: $\mathrm{oid}(c) = \{\underline{c}_1, \underline{c}_2, ...\}$. Usually single letters combined with increasing indexes are used to derive the objects identifiers. The examples of object names definitions are implemented as the class operations: *develsname()* for generics (see the chapter 3.1) and *dispname()* for CC components (see the chapter 3.2). Please note that there are no different objects with the same name assigned. The domain of the class $c \in \mathrm{CLASS}$ encompasses all objects of the class c and all objects of its child classes:

$$I_{\mathrm{CLASS}}(c) = \bigcup \{\mathrm{oid}(c') \mid c' \in \mathrm{CLASS} \wedge (c' \prec c \vee c' = c)\}.$$

Please note that in this monograph two domains representing developers' specification means are created: $I_{\mathrm{SICLASS}}(\mathit{Generic})$ and $I_{\mathrm{SICLASS}}(\mathit{CCSecComponent})$.

The developers use objects of the `GenItem`, `FunComp`, `AssComp` classes, i.e. generics and/or components that have identifiers issued respectively by the *develsname()* or *dispname()* operations. Their parent classes have rather taxonomical meaning. The assumed naming convention is a little complicated but it complies with the syntax of generics and components used by developers.

The generalization hierarchy implies a subset relation on the semantic domain of classes:

$\forall c_1, c_2 \in \mathrm{CLASS}: c_1 \prec c_2 \Rightarrow I(c_1) \subseteq I(c_2).$

For each of the above defined full descriptors $\mathrm{FD}_{\mathrm{GenItem}}$, $\mathrm{FD}_{\mathrm{FunComp}}$, $\mathrm{FD}_{\mathrm{AssComp}}$ (with respect to their generalization hierarchies) the following subset relations exist:

$I(\mathtt{GenItem}) \subseteq I(\mathit{GenFamily}) \subseteq I(\mathit{GenGroup}) \subseteq I(\mathit{Generic}),$

$I(\mathtt{FunComp}) \subseteq I(\mathit{FunCompFamily}). \subseteq I(\mathit{FunSecClass}) \subseteq I(\mathit{CCSecComponent}),$

$I(\mathtt{AssComp}) \subseteq I(\mathit{AssCompFamily}) \subseteq I(\mathit{AssSecClass}) \subseteq I(\mathit{CCSecComponent}).$

Some comments ought to be added concerning links implied by two kinds of associations: $\mathrm{GenParAssoc} \subseteq \mathrm{SIASSOC} \subseteq \mathrm{ASSOC}$ and $\mathrm{SecAssoc} \subseteq \mathrm{SIASSOC} \subseteq \mathrm{ASSOC}$. Both are defined on the group level expressing common relationships between security items concerning the parameterization and/or mapping. Links implied by these associations concern the objects of the GenItem, FunComp, AssComp classes.

With respect to parameterization:

$\forall\ as_{\mathrm{Param4Parameterized}} \in \mathrm{GenParAssoc},$

$as_{\mathrm{Param4Parameterized}} \mapsto \langle \mathit{Parameterized}, \mathit{Parameter} \rangle :$

$\mathtt{GenItem} \prec \mathit{Parameterized} \vee \mathtt{GenItem} \prec \mathit{Parameter}.$

With respect to mapping:

$\forall\ as_{\mathrm{Mapping}} \in \mathrm{SecAssoc}, as_{\mathrm{Mapping}} \mapsto \langle \mathit{IsCovered}, \mathit{Covers} \rangle :$

$(\mathtt{GenItem} \prec \mathit{IsCovered} \vee \mathtt{GenItem} \prec \mathit{Covers}) \wedge$

$(\mathtt{FunComp} \prec \mathit{IsCovered} \vee \mathtt{FunComp} \prec \mathit{Covers}) \wedge \mathtt{AssComp} \prec \mathit{Covers}.$

The association $as_{\mathrm{Param4Parameterized}} \in \mathrm{GenParAssoc}$ with
$\mathrm{associates}(as_{\mathrm{Param4Parameterized}}) = \langle c_1, \ldots, c_n \rangle$, where $\forall\ i \in \{1, \ldots, n\}\ c_i \in \mathtt{GenItem}$,
is interpreted as the Cartesian product of the set of object identifiers of the participating classes:

$I_{\mathrm{GenParAssoc}}(as_{\mathrm{Param4Parameterized}}) = I_{\mathrm{SiCLASS}}(c_1) \times \ldots \times I_{\mathrm{SiCLASS}}(c_n),$

where a link denoting a connection between objects is an element

$l_{as_{\mathrm{Param4Parameterized}}} \in I_{\mathrm{GenParAssoc}}(as_{\mathrm{Param4Parameterized}}).$

The objects linked by mapping can be expressed in a similar way though in this case not only generics are used but also components.

The association $as_{\mathrm{Mapping}} \in \mathrm{SecAssoc}$ with $\mathrm{associates}(as_{\mathrm{Mapping}}) = \langle c_1, \ldots, c_n \rangle$,
where $\forall\ i \in \{1, \ldots, n\}\ (c_i \in \mathtt{GenItem}) \vee (c_i \in \mathtt{FunComp}) \vee (c_i \in \mathtt{AssComp})$ is interpreted as the Cartesian product of the set of object identifiers of the participating classes:

$$I_{GenParAssoc}(as_{Mapping}) = I_{SiCLASS}(c_1) \times \ldots \times I_{SiCLASS}(c_n),$$

where a link denoting a connection between objects is an element

$$l_{as_{Mapping}} \in I_{SecAssoc}(as_{Mapping}).$$

Definition 3.63: System state of the $\mathcal{M}_{Si}$.

The system state of the model $\mathcal{M}_{Si}$ is a structure:

$$\sigma(\mathcal{M}_{Si}) = (\sigma_{SiCLASS}, \sigma_{ATT}, \sigma_{SiASSOC}), \text{ where:}$$

i. The finite sets $\sigma_{SiCLASS}(c)$ include all objects of the class $c \in SiCLASS$ existing in the system state:

$$\sigma_{SiCLASS}(c) \subset oid(c),$$

ii. Functions σ_{ATT} assign attribute values to each object:

$$\sigma_{ATT}(a): \sigma_{SiCLASS}(c) \rightarrow I(t) \text{ for each } a: t_c \rightarrow t \in ATT^{*}{}_{c},$$

iii. The finite sets $\sigma_{SiCLASS}$ contain links (satisfying multiplicities) connecting objects:

$$\forall\, as \in SiASSOC, \sigma_{SiASSOC}(as) \subset I_{SiASSOC}(as).$$

□

4. Capturing the features of an IT security-related product or system

4.1. General product or system presentation according to the standard

The introductory parts of the ST or PP documents require a concise presentation of the developed product (or system). It will be used for a quick product review by the people who select products to meet specific needs of the organization. Usually, they are business or IT managers. It is obvious that a short presentation will be developed to get the ST or PP documents ready for the evaluation as well.

The presented IT Security Development Framework assumes that the first step of capturing the security-relevant system characteristics is taken during the internal product or system description and the TOE description developed on this basis, required by the ST or PP documents.

This should be concentrated around the security attributes, i.e. integrity, confidentiality and availability. This is a commonly used approach for security needs or risk analysis. During the development of the product models, the separation-of-concern principle is recommended.

The designing process of the security-related product, corresponding to the elaboration of Security Target documentation, begins with preparing Security Target [38] introduction, containing rather trivial information, such as ST identification, general description, PP claims and TOE description that should be worked out. The TOE description, containing a concise product presentation for consumers, includes the following kinds of data (Fig. 4.1):

- General TOE functionality – not limited to the security features unless the TOE is a special-purpose security product,
- Product or system type, like database, encrypting device, firewall, etc.
- Logical boundaries – what is there in the TOE and what is not, in terms of security features and services,
- Physical boundaries – what is there in the TOE and what is not, in terms of hardware/software components or modules,
- TOE operational environment.

Generally, it should be known how the TOE works within its environment and what it consists of, using logical or physical terms. This is the reference point for further stages of the whole TOE development process, deciding about its cost and quality. The TOE may be software, hardware or a system. Its design analysis is needed. How deep it should be depends mostly on the assumed EAL level. Sometimes full technical documentation exists, sometimes it is created concurrently with the security documentation (PP/ST).

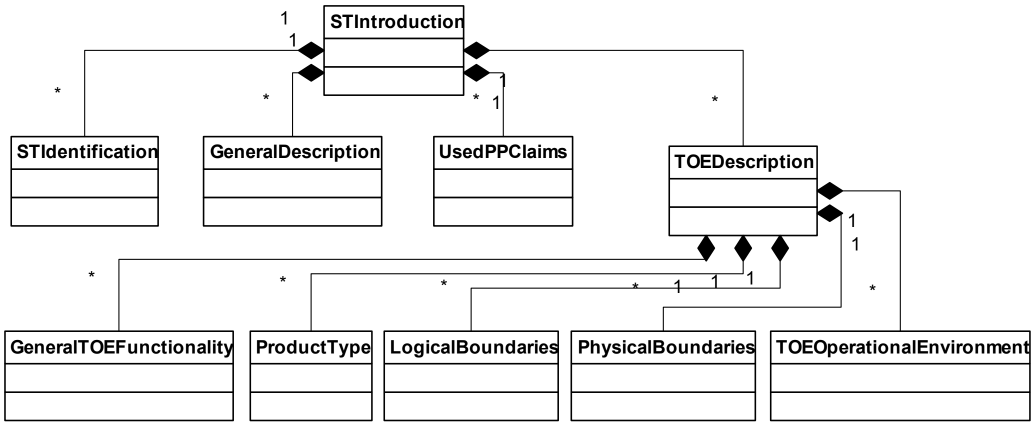


Fig. 4.1. Security Target introduction elements (Common Criteria)
Rys. 4.1. Elementy wprowadzenia do zadania zabezpieczeń (Wspólne kryteria)

The following sections of this chapters contain the concept how to semiformally elaborate the ST introduction specification being the starting point of the whole ST development.

4.2. Modelling the basic features of the product or system (BCL)

The required product or system descriptors (Fig. 4.1) can usually be prepared straight on the basis of technical documentation, although a more convenient and more formal approach is proposed there. For any developed product or system, a rough, three-level auxiliary model is created (i.e. PM model – Fig. 2.3), being input for the ST or PP model.

According to the presented methodology, the actual ST introduction model elaboration is preceded by the preliminary stage – the BCL – Business/consumer level model workout. The left part of the Fig. 4.2 contains the BCL classes, and the right part – the ST introduction classes, elaborated on this basis.

The elaboration of the BCL model is shown in the Fig. 4.3. Please note the states (grey colour) concerning the control of the IT security development process on entry and on exit of the BCL development stage.

This elaboration begins (the first white state) with a short presentation of the purpose of the product or system, answering the question: “What tasks will be performed using the product or system?” Next, the kind of data – processed, stored or transmitted within the TOE, is described. At minimum, the following kinds of data should be considered: financial information (credit card numbers, bank account information, reports), personal data, data dealing with health, technological data, research data, organization-strategic data, or other specified sensitive data.

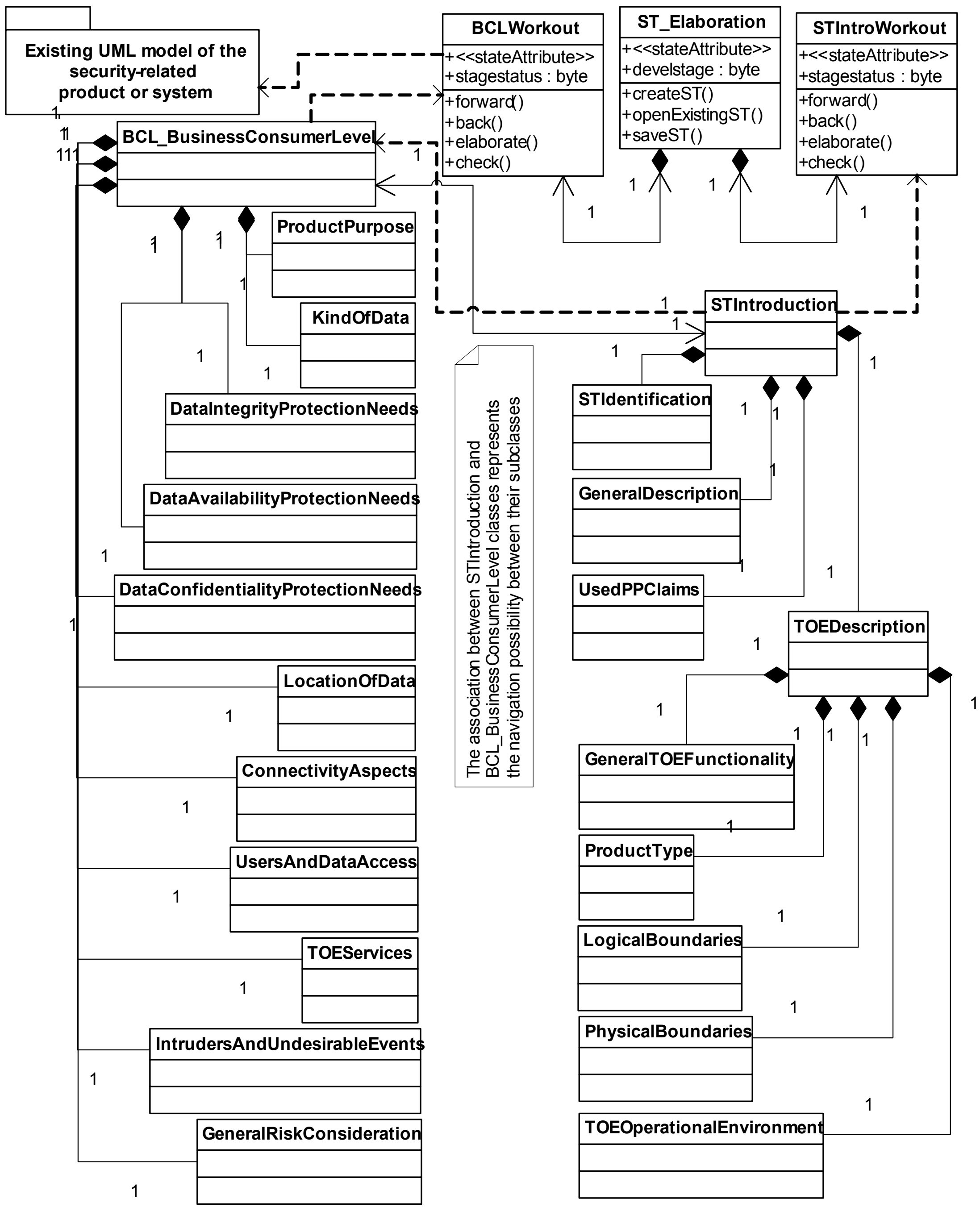


Fig. 4.2. The classes participating in the BCL and the Security Target introduction models elaboration

Rys. 4.2. Klasy wykorzystywane podczas tworzenia modelu BCL oraz modelu wprowadzenia do zadania zabezpieczeń

Three additional items deal with identifying the scope of the protection needs. The effects of the loss, fabrication or modification of the information, should be considered with respect to the data integrity. It should be noted how critical the data accuracy for the TOE users is. For the data availability, the effects of the denial or delay of the information should be considered. It ought to be analyzed how critical the data availability can be. In the scope of the data confidentiality needs, the following questions should be clear:

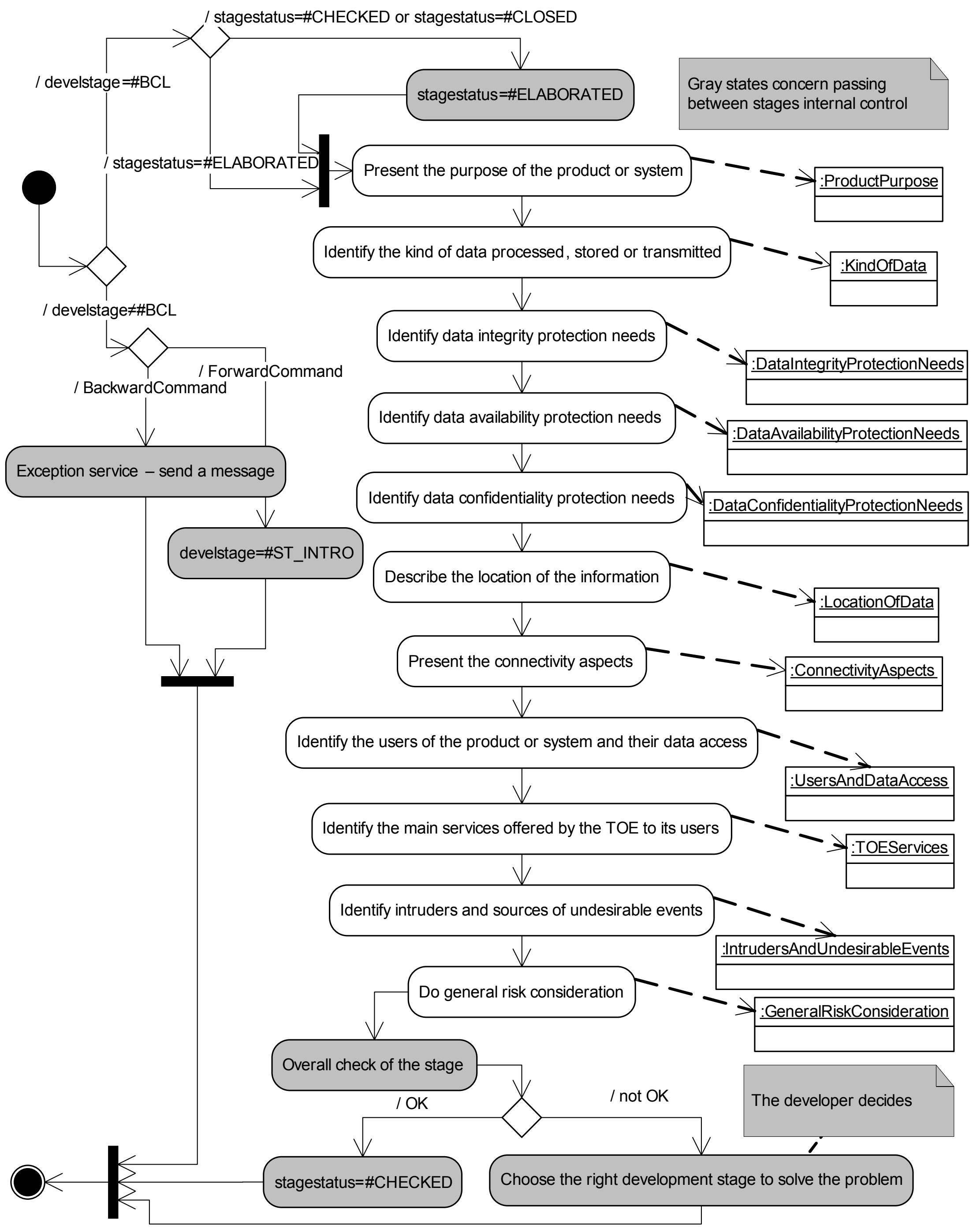

/ stagestatus=#CHECKED or stagestatus=#CLOSED
/ develstage=#BCL
stagestatus=#ELABORATED
Gray states concern passing between stages internal control
/ stagestatus=#ELABORATED
Present the purpose of the product or system
:ProductPurpose
Identify the kind of data processed, stored or transmitted
:KindOfData
/ develstage≠#BCL
Identify data integrity protection needs
:DataIntegrityProtectionNeeds
/ ForwardCommand
/ BackwardCommand
Identify data availability protection needs
:DataAvailabilityProtectionNeeds
Exception service – send a message
Identify data confidentiality protection needs
:DataConfidentialityProtectionNeeds
develstage=#ST_INTRO
Describe the location of the information
:LocationOfData
Present the connectivity aspects
:ConnectivityAspects
Identify the users of the product or system and their data access
:UsersAndDataAccess
Identify the main services offered by the TOE to its users
:TOEServices
Identify intruders and sources of undesirable events
:IntrudersAndUndesirableEvents
Do general risk consideration
:GeneralRiskConsideration
Overall check of the stage
/ OK
/ not OK
The developer decides
stagestatus=#CHECKED
Choose the right development stage to solve the problem

Fig. 4.3. The BCL model elaboration
Rys. 4.3. Wypracowanie modelu typu BCL

- What kind of data can be considered as sensitive data?
- What are the effects of disclosing the information to unauthorized parties?

- How critical is data confidentiality?

The next two stages concern the location of the information and connectivity aspects. The following important questions have to be taken into consideration:

- Where are the information assets protected by the TOE?
- Are they inside the product or system, or outside?
- Is the TOE a closed IT product or a distributed IT system?
- What are other modules or subsystems which create an IT environment for a given system?
- What are the systems interconnections?

The next step encompasses a short presentation of the product or system users (main group of the model actors) and basic rules of the data access:

- Who will be the users of the product or system?
- Who has access to the system and its data?
- Have all individuals access to all data on the system or will different levels of access be considered?

The key issue is always the TOE services, because they are the reason for developing the product or system. The main services offered by the TOE to its actors should be identified. On this basis the use case models will be created.

Not only the legal users will affect the TOE. The intruders and other sources of undesirable events that may cause impacts should be considered too:

- human entities may disturb the TOE usage (trusted insiders, authorized users who may unintentionally invoke errors or accidents, external intruders);
- non-human causes of undesirable events may disturb the TOE work.

They will be used as actors during the risk model refinement at the TOE environment elaboration stage.

Finally, the general risk consideration is recommended, encompassing:

- a simple trade-off analysis dealing with the assumed assurance and cost of measures,
- analyzing the warranty of the cost of intended measures to achieve information security at the desired level.

The above mentioned trade-off analysis should evaluate the consequences of each of the following scenarios:

- protection (or over protection) of the product or system with expensive measures that can also affect its performance,
- implementing a minimal set of measures that leave the system almost open to either outside intrusion or internal attack.

4.3. Security Target introductory part elaboration

The information captured during the BCL model (Fig. 4.2) workout is general but precise enough to develop an ST introduction (Fig. 4.2). This is shown in the Fig. 4.4. The states marked grey concern the control of the IT security development process.

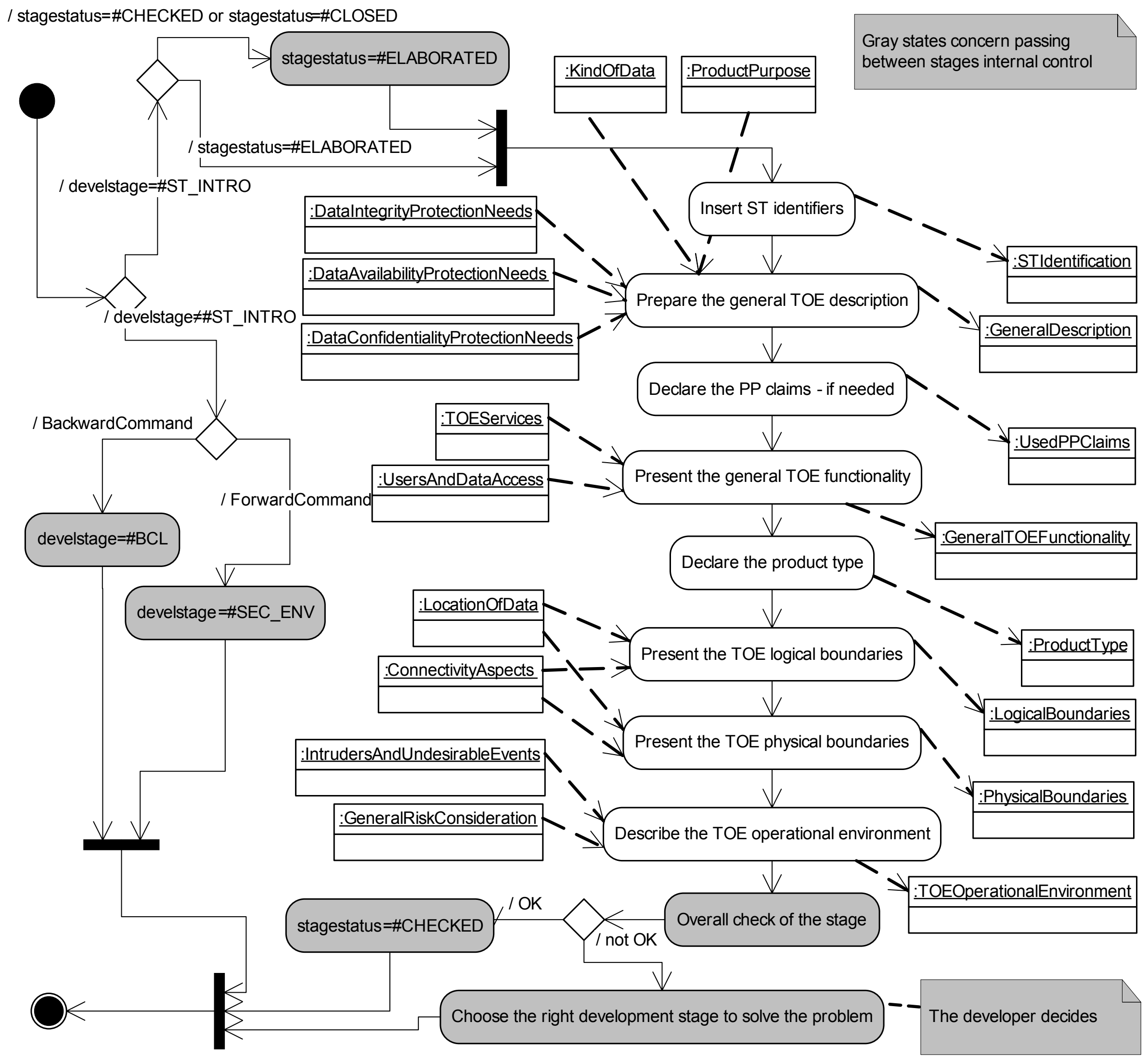

/ stagestatus=#CHECKED or stagestatus=#CLOSED
stagestatus=#ELABORATED
:KindOfData
:ProductPurpose
Gray states concern passing between stages internal control
/ stagestatus=#ELABORATED
/ develstage=#ST_INTRO
Insert ST identifiers
:DataIntegrityProtectionNeeds
:STIdentification
:DataAvailabilityProtectionNeeds
Prepare the general TOE description
/ develstage≠#ST_INTRO
:DataConfidentialityProtectionNeeds
:GeneralDescription
Declare the PP claims - if needed
/ BackwardCommand
:TOEServices
:UsedPPClaims
:UsersAndDataAccess
Present the general TOE functionality
/ ForwardCommand
develstage=#BCL
:GeneralTOEFunctionality
Declare the product type
develstage=#SEC_ENV
:LocationOfData
:ProductType
Present the TOE logical boundaries
:ConnectivityAspects
:LogicalBoundaries
Present the TOE physical boundaries
:IntrudersAndUndesirableEvents
:PhysicalBoundaries
:GeneralRiskConsideration
Describe the TOE operational environment
:TOEOperationalEnvironment
/ OK
stagestatus=#CHECKED
Overall check of the stage
/ not OK
Choose the right development stage to solve the problem
The developer decides

Fig. 4.4. ST introduction workout
Rys. 4.4. Wypracowanie modelu wprowadzenia do ST

The sampled information is textual and informal, dedicated mainly to the product or system recognition on the market. The BCL model is precise enough to specify an ST introduction, but it is not enough as a basis for future SM model development. On the basis of the BCL model, a more detailed UAL model is developed.

4.4. Modelling structural and behavioural aspects (UAL)

The UAL model, generally optional but recommended, should present a product or system at an appropriate level to develop PP/ST, particularly it can be helpful for the BCL workout on its basis. Please note that it depends strongly on the subject, i.e. the product or system can be presented there at a very general level. It is assumed that given UML specifications exist, like those presented in the Fig. 4.5. The UAL models can be created using the existing UML methodologies, e.g. those mentioned in chapter 1, like the UMLsec.

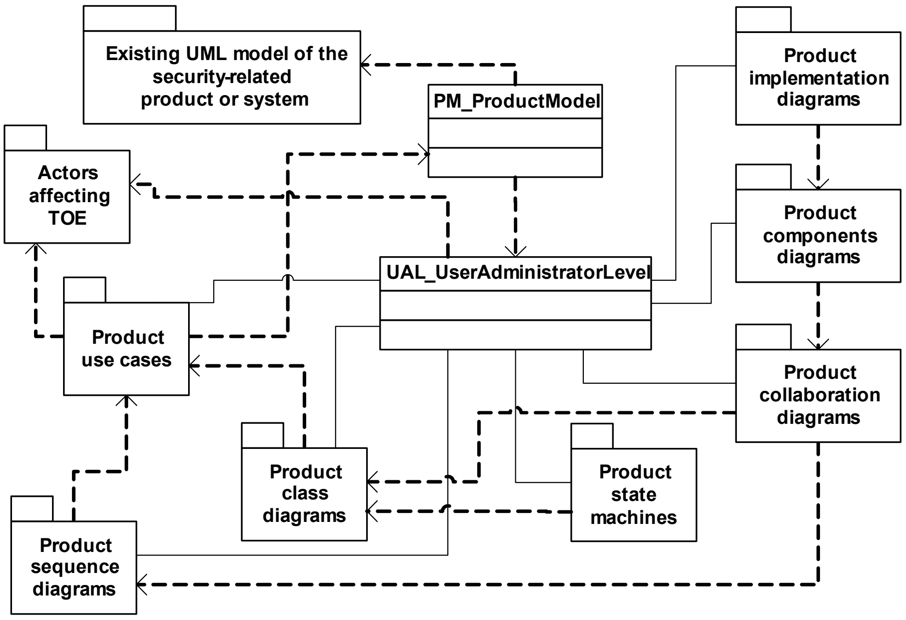

Existing UML model of the security-related product or system
PM_ProductModel
Product implementation diagrams
Actors affecting TOE
Product components diagrams
UAL_UserAdministratorLevel
Product use cases
Product collaboration diagrams
Product class diagrams
Product state machines
Product sequence diagrams

Fig. 4.5. The UAL model
Rys. 4.5. Model typu UAL

The presented method of the TOE description workout assumes the UML approach. It is based on the use cases and actors concept. On this basis the collaborations and interaction diagrams are worked out. This detail level is suitable and enough for capturing the basic TOE features, especially for COTS. These features should be contained in the TOE description, i.e. basic functionality, logical and physical boundaries, basic modules and the TOE environment. A more detailed model can be elaborated when needed (higher EALs, the need of investigating specific behavioural aspects or performing a detailed risk analysis).

The TOE, as an IT product, can be considered as a black box with certain functions provided for the outside environment. The actors, i.e. users, administrators, attackers, etc., can stimulate operations on the TOE. It should be noted that non-security features are considered unless the TOE is a special-purpose security product (Fig. 4.6). The actor works outside the TOE (i.e. system) boundaries.

The TOE is a rather complex system, having a set of use cases, bringing expected results to the actors. For example, they expect to perform the specified set of operations on the TOE successfully.

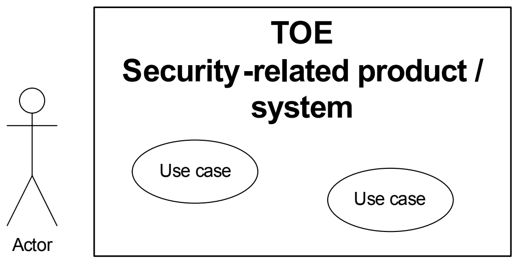


Fig. 4.6. TOE use case concept
Rys. 4.6. Koncepcja przypadków użycia dla przedmiotu oceny (TOE)

Let us consider the TOE as the source of services provided for actors, bringing them the expected results. The actors act according to the scenarios encompassed by the use cases.

There are many use cases and many actors too. Some of them are legal, some may be illegal and even malicious. All main types of actors and their generalizations are shown in the Fig. 4.7. Please note the `SgrGeneric` elements representing them.

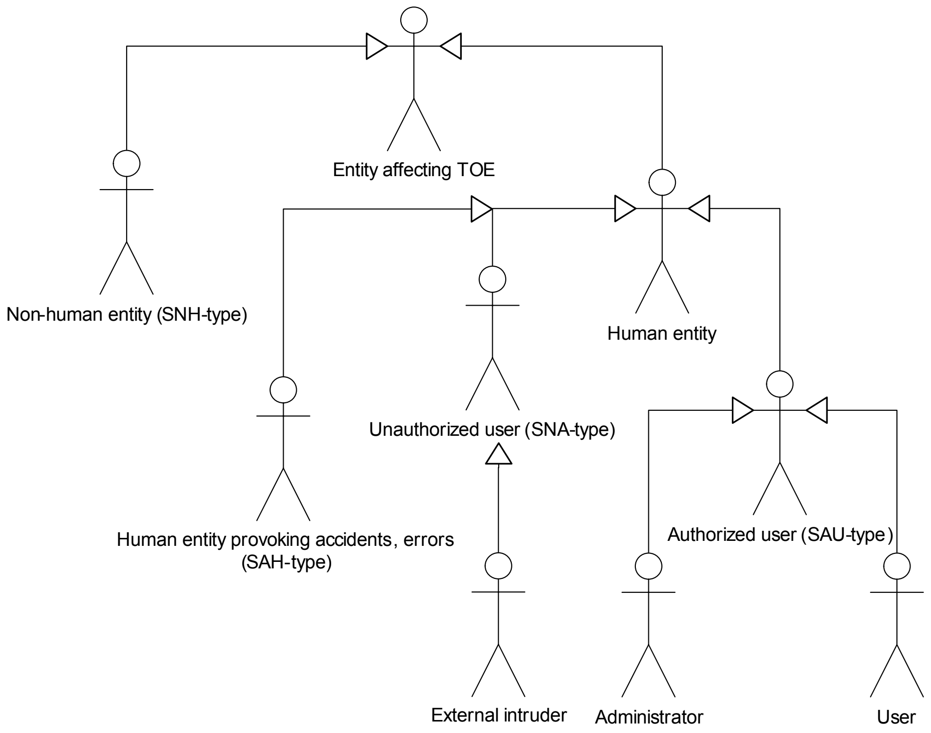

Entity affecting TOE
Non-human entity (SNH-type)
Human entity
Unauthorized user (SNA-type)
Human entity provoking accidents, errors (SAH-type)
Authorized user (SAU-type)
External intruder
Administrator
User

Fig. 4.7. Main actor types affecting the TOE
Rys. 4.7. Główne rodzaje aktorów oddziaływujących na przedmiot oceny (TOE)

Please note that the presented method not only considers human actors but also introduces non-human ones, expressing different sources of undesirable events within the TOE environment that should be taken into consideration during a future risk analysis. The actors correspond to different subject-group generics (in the CC called "active entities") [13].

While identifying the TOE functionality, many use cases should be considered. Some of them provide legal actors with desired results, some do not. While preparing the TOE description, only legal actors and TOE services offered for them are taken into consideration.

It should be emphasized that there are both human and non-human actors, like intruders, reckless people, force majeures, natural catastrophes, etc., negatively influencing the system and bringing undesirable results. All of them will be considered later while carrying out the risk analysis.

Every use case is generalization of its scenarios (i.e. given variants of its development) representing a set of interactions or events bringing the expected and well defined result to the actor.

In the UML a given use case can include (<<uses>>) or be extended (<<extends>>) by other use cases. It allows to order actions on the TOE and group them into the subsets of use cases (Fig. 4.8).

Use cases have their own attributes, as internal variables, and the operations representing internal actions.

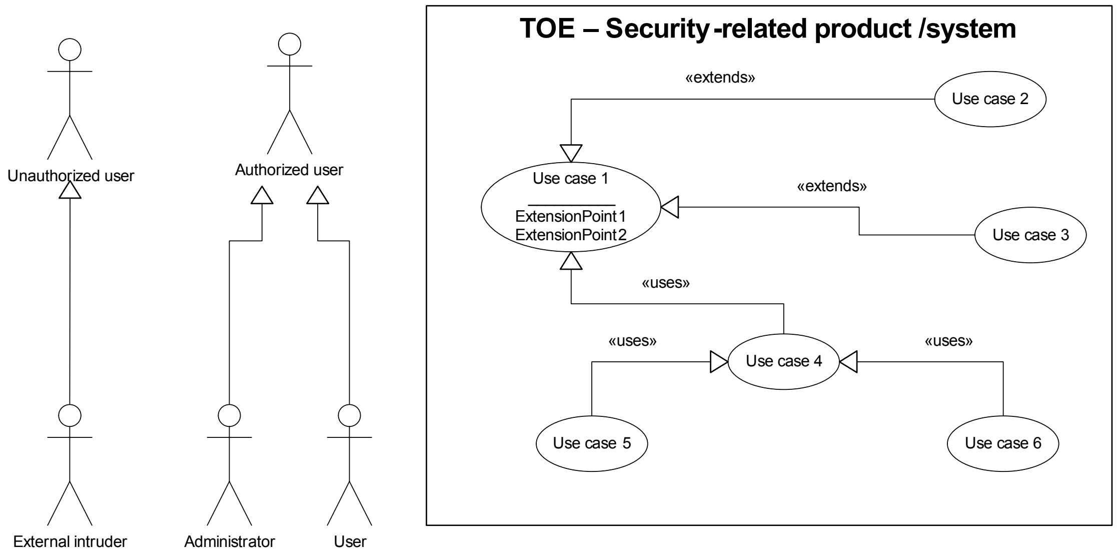


Fig. 4.8. The set of actors interacting with the system and the set of use cases
Rys. 4.8. Zbiór aktorów prowadzących interakcje z systemem oraz zbiór przypadków użycia

There are two main problems during security-related product identification. The first one is to specify the TOE functionality, based on the expected behaviour. These functions must be performed by the users in a secure way. For this reason all potential causes that may disturb the TOE operations have to be detected too. This is the second issue to solve. It will be considered later using the risk analysis approach during the TOE security environment elaboration.

In the Fig. 4.9 actors, use cases and their associations were shown as an example. There are two actors (user, administrator), each belonging to two associations. Use case 1 has two extension points suitable for Use cases 2 and 3, representing optional variants of Use case 1. Use case 4, with its Use cases 5 and 6, are all included in Use case 1, and they are never stimulated by the actors. The extension point is a place to take the decision if an extended use case (as the variant) should be processed.

Please note that the actors reside outside the TOE – within its environment, but according to the CC functional paradigm [39] each of them is represented by a subject belonging to the TOE. Notes with requirements or remarks for future consideration can be added to the diagram. Sometimes minor additional use cases, performing internal, not actor's tasks, may be attached (monitoring, self testing, etc).

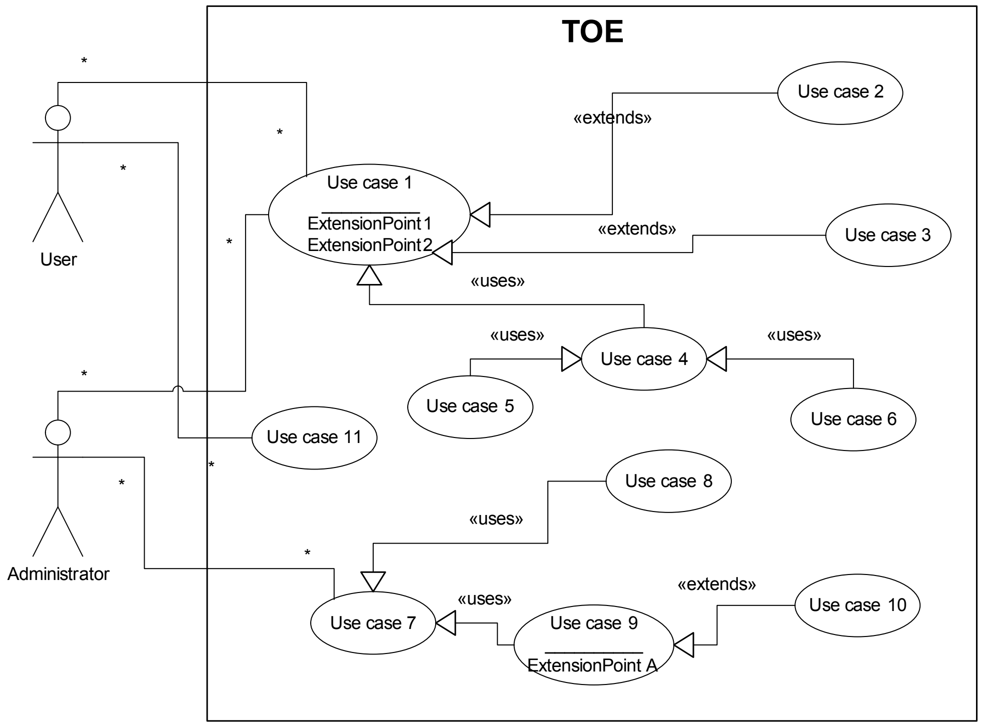


Fig. 4.9. Use case diagram
Rys. 4.9. Diagram przypadków użycia

Every use case should be focused on a single important issue (expected results). Its set of actions can be expressed verbally by a pseudo-code, or better, by an activity diagram or even by a state machine. Use cases stimulated by actors will be called main use cases. Each of them corresponds to one of the TOE providing services.

In the Fig. 4.10 an example of an activity diagram for Use case 7 (Fig. 4.9) is shown. The number of activities encompassed by a given use case depends on the TOE features. In the extension point, placed within Use case 9, the decision dealing with the inclusion of Use case 10 was made, expressing one of the possible processing variants of Use case 7. Use case 8 is always included but it is not stimulated by any actor.

Elementary functionality, called there the TOE service, corresponding to any main use case processing, can be now identified and added do the General TOE functionality within the ST data structure.

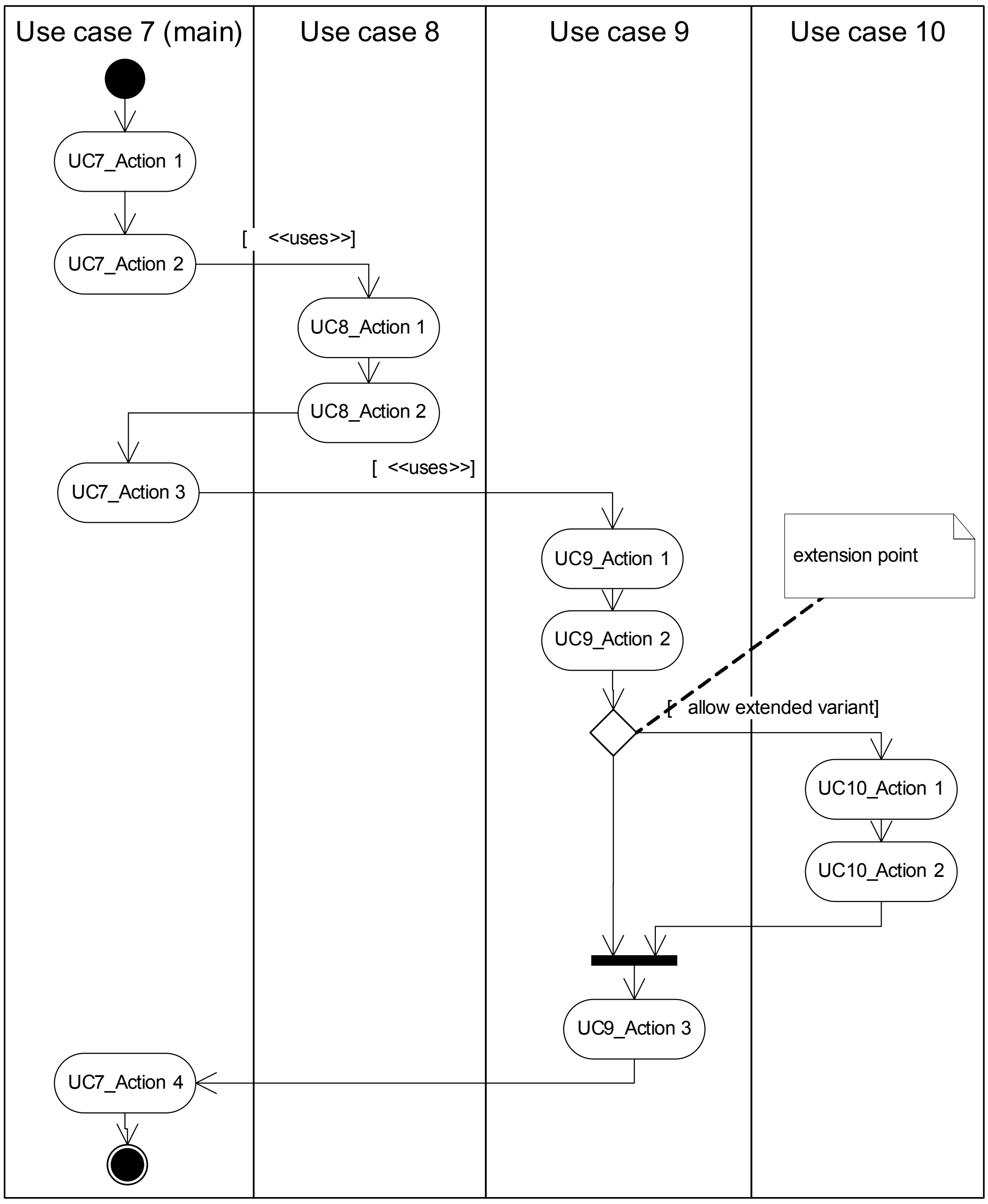


Fig. 4.10. Activity diagram for a use case – example
Rys. 4.10. Diagram czynności odpowiadający przypadkowi użycia – przykład

After preparing the use case diagram the next step is possible on this basis – the development of corresponding collaborations.

Each collaboration, being the development of a use case, has two aspects, strongly dependent on the TOE nature:

- structural (static), presenting collaborating classes, interfaces, and elements in the form of a class diagram,
- behavioural (dynamic), represented by interaction diagrams.

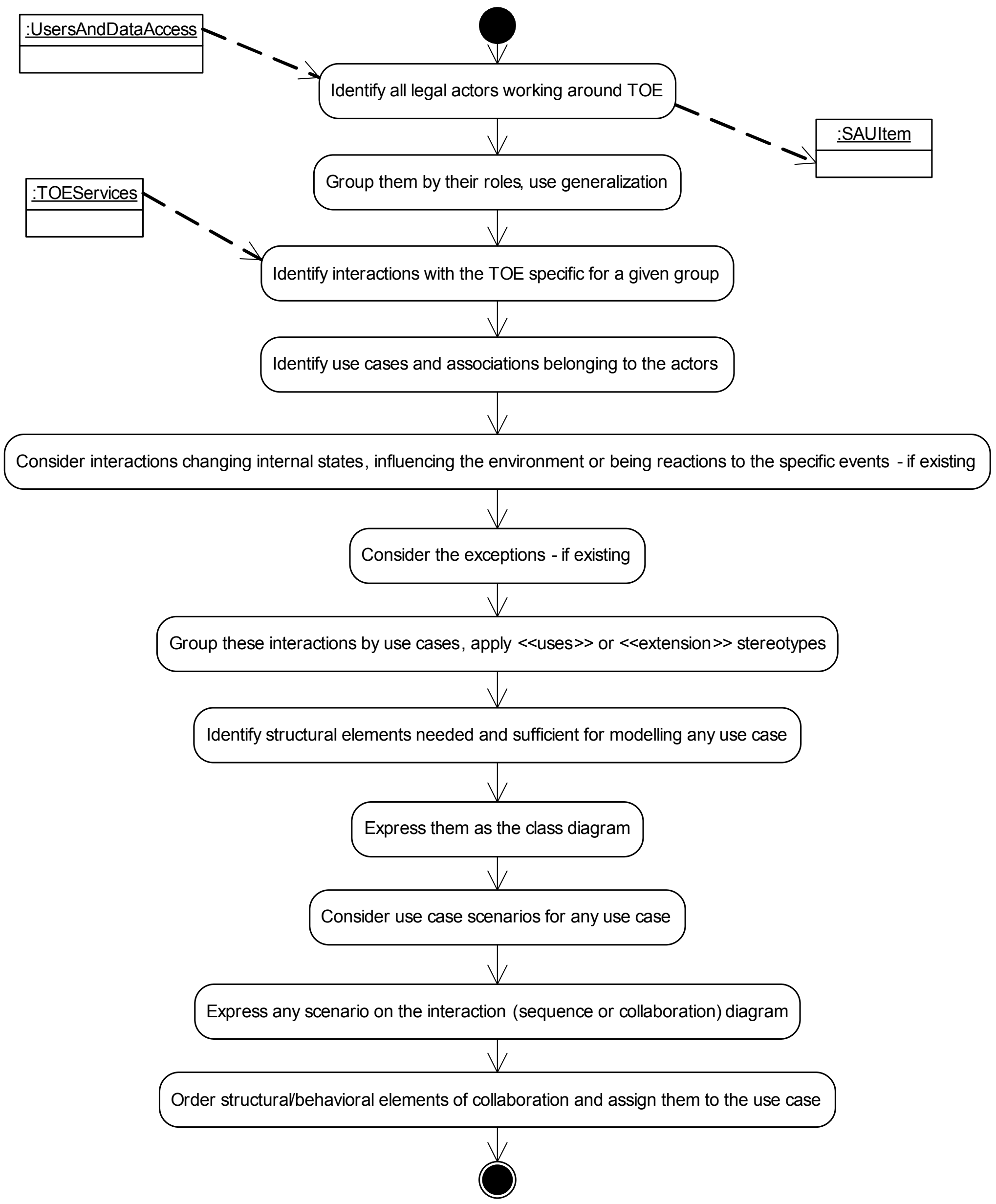

:UsersAndDataAccess
Identify all legal actors working around TOE
:SAUItem
Group them by their roles, use generalization
:TOEServices
Identify interactions with the TOE specific for a given group
Identify use cases and associations belonging to the actors
Consider interactions changing internal states, influencing the environment or being reactions to the specific events - if existing
Consider the exceptions - if existing
Group these interactions by use cases, apply <<uses>> or <<extension>> stereotypes
Identify structural elements needed and sufficient for modelling any use case
Express them as the class diagram
Consider use case scenarios for any use case
Express any scenario on the interaction (sequence or collaboration) diagram
Order structural/behavioral elements of collaboration and assign them to the use case

Fig. 4.11. The UAL model refinement
Rys. 4.11. Uszczegółowienie modelu typu UAL

The whole modelling process, providing the UAL model (Fig. 4.5), starting from use cases specification to identifying internal elements of collaboration, is summarized by means of the activity diagram presented in the Fig. 4.11.

Please note that all use cases mentioned there deal with the TOE intentional usage. The UAL model is based on the main use cases corresponding to the TOE-offered services. It expresses interactions between the TOE and the actors representing legal users, acting with the TOE environment. In the same case additional groups of actors, representing illegal activities and undesirable events, will be added during the next stage of the IT security development process – the threat analysis. The third possibility, not discussed there, are typical use case diagrams to specify users' (i.e. IT security – developers, officers, sponsors, and IT administrators) interactions with the computer-aided tool developed on the basis of the methodology presented there.

The collaborations specify basic TOE elements, relations and interactions between them, presenting global behaviours of the system and all its actors.

A general example of collaboration with its two aspects is presented in the Fig. 4.12. These elements ought to correspond to the modelled TOE.

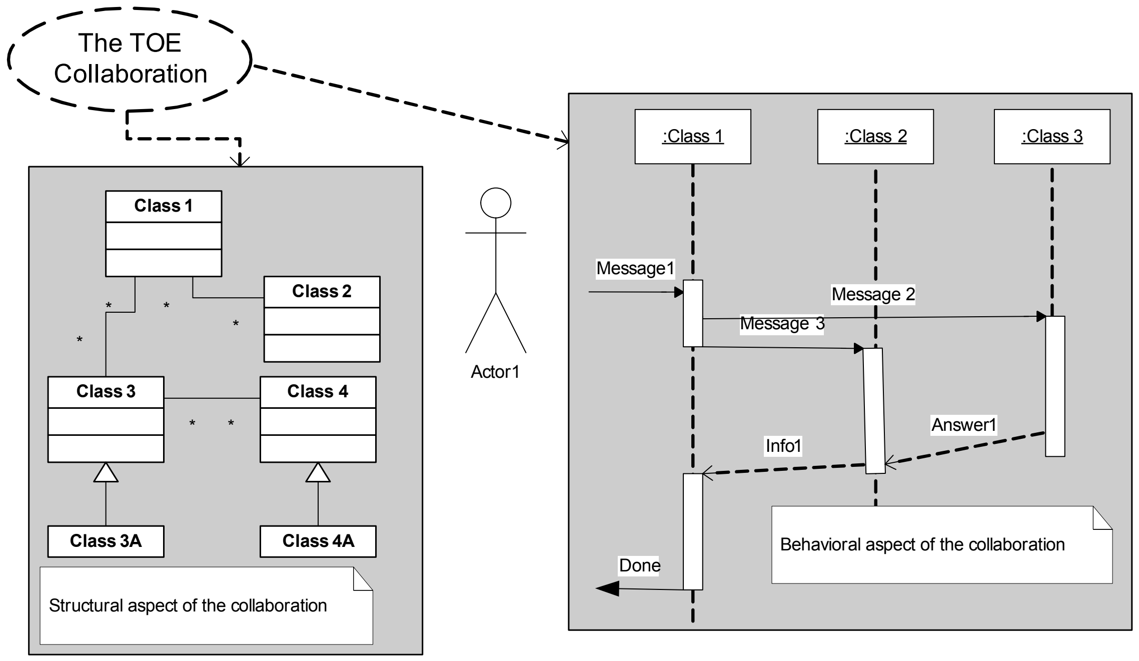


Fig. 4.12. Structural and behavioural aspects of the collaboration
Rys. 4.12. Strukturalne i behawioralne aspekty kooperacji

The modelling process allows to refine the key parts of the TOE description, if higher precision offered by this UML approach is needed:

- General TOE functionality – by using all TOE services dealing with main use cases and behavioural aspects of the collaboration,
- Logical boundaries – corresponding to a class diagram, expressing structural aspects of the collaboration,

- Physical boundaries – using a deployment diagram,
- TOE operational environment – using a deployment diagram.

Deployment diagrams present physical components and nodes that belong to the TOE but also work outside the TOE – within its operational environment (Fig. 4.13).

The level of details of all above diagrams and schemes strongly depends on the declared EAL level.

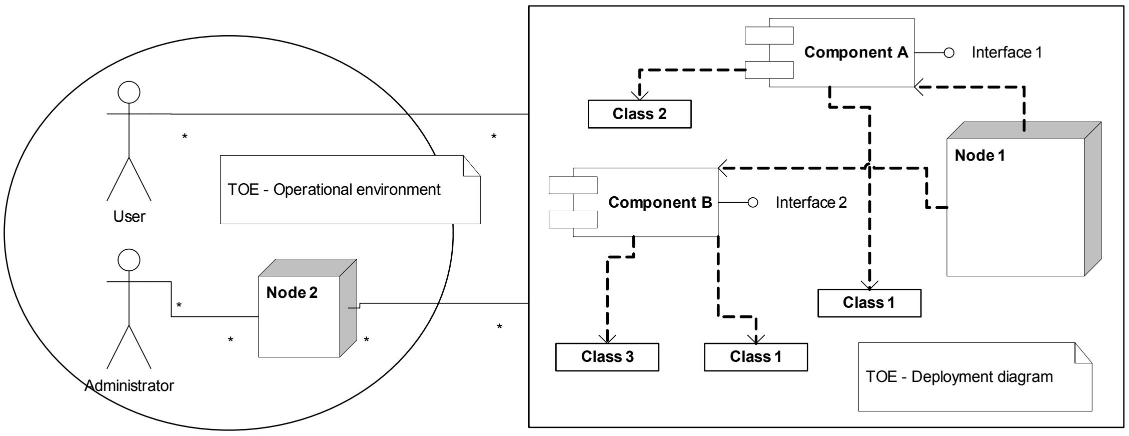


Fig. 4.13. TOE physical boundaries and operational environment
Rys. 4.13. Fizyczne granice przedmiotu oceny (TOE) oraz jego środowisko eksploatacji

The summarizing scheme of the ST introduction elaboration was presented in the Fig. 4.14. It corresponds to the first ST development stage.

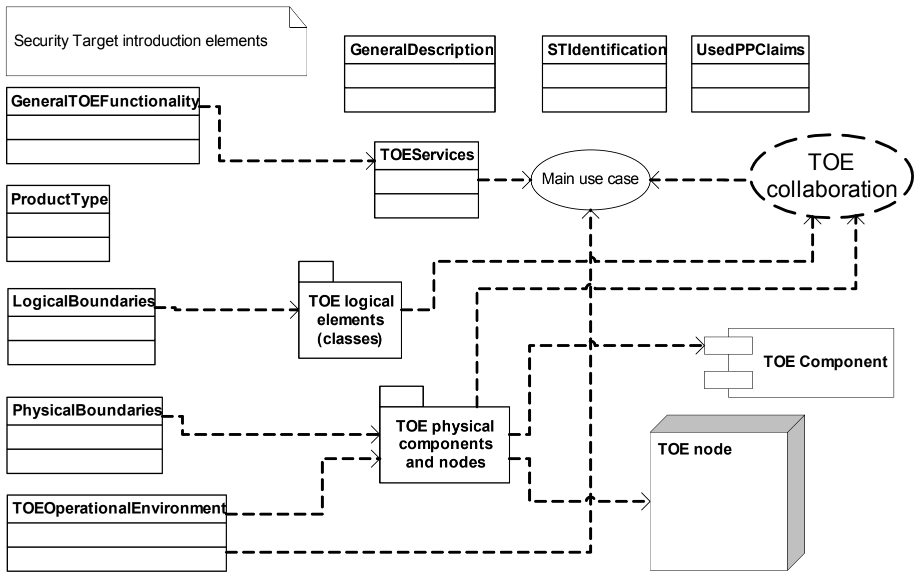


Fig. 4.14. ST introduction collaboration
Rys. 4.14. Kooperacja odpowiadająca wprowadzeniu do zadania zabezpieczeń

General TOE functionality expresses the TOE services identified on the use case diagrams. Every TOE service will be considered as a pair: an authorized user and elementary functionality offered to him/her by the TOE. Logical boundaries encompass the TOE structural elements developed on the use case diagram analysis. The package containing identified physical elements, being the final implementation of the TOE collaboration, is used for physical boundaries specification and operational environment. The latter depends also on the use case diagram, which represents, among other things, actors acting outside the TOE.

The elements of the ST introduction are on a very general level. All of them should be refined and supplemented during next development stages.

4.5. Compatibility with the UMLsec

The mentioned above UMLsec formal methodology (and the formal verification tools) [72] can be useful for modelling some kinds of IT products of higher EALs. Such models, used there as input models, are expressed by the `EUM_EntryUMLmodel` (Fig. 2.3, Fig. 2.6). The problem is how to use and interpret the UMLsec-type model on entry of the IT security development process, where the `BCL_BusinessConsumerLevel` model is created.

Continuing the UMLsec presentation (Section 2.1.2), this transformation will be shown very briefly, explaining the selected UMLsec issues, as well as some issues concerning cryptographic protocols that quite frequent. The UMLsec and the methodologies presented in the monograph are complementary regarding the area of application:

- the UMLsec can be used for the precise modelling of IT products (rather products, not systems) allowing to achieve higher EALs for which a formal approach is required,
- the presented Common Criteria compliant IT security development methodology can be used concurrently, issuing documentation required for the IT product evaluation and certification.

Ensuring their compatibility allows also to achieve their mutual support.

The entire IT product or (sub)system is modelled as a set of stereotyped objects and/or (sub)systems that exchange messages with the use of the UML diagrams. The work of each object or system component O can be expressed by the UML machine processing input queues $inQu_O$ to the output queues $outQu_O$. The UML machine $[[C]]$ represents the behaviour of all such objects or components C directly contained in C.

$Events$ is a set of messages exchanged between objects and/or systems while they communicate with each other. Receiving such a message is called $event$. The $message$ consists of a message name $\in MsgNm$ and message arguments. The message name may be preceded by object or system instance names $O \in UMNames$. The message arguments belong to Exp which is a set of expressions.

Sending a message $msg=op(exp_1, \ , exp_i, \ , exp_n)$ from an object or subsystem instance S to an object or subsystem instance R can be described as follows in a concise way, where $msg \in Events$, $exp_i \in Exp$, and op is an operation:

i. S places the message $R.msg$ into its $outQu_S$,

ii. A scheduler removes the $R.msg$ from the $outQu_S$ to the $inQu_R$,

iii. R removes $R.msg$ from the $inQu_R$ and starts its processing.

This behaviour is usually presented on sequence diagrams. An execution of a UML subsystem S, which may contain the components $C_1,..., C_j,..., C_m$, is expressed by the sequence of states and the associated bags of input and output messages of $[[S]]$.

In [72] the common cryptographic terms are used, which are presented in the Appendix C. Please note that the UMLsec uses common name space for the product or system models and for the applied cryptography. The following sets represent: $Keys$ – cryptographic keys, Var – variables and $Data$ – data values, where:

- $Keys \cap Var \cap Data = \varnothing$, and
- $UMNames \cup MsgNm \cup Secrets \subseteq Data$.

The UMLsec introduces its own definitions [72] of security properties that ought to be presented in a concise way, i.e. confidentiality (there: secrecy), integrity, authenticity, freshness, and secure data flow – all considered with respect to the attacker (here: adversary) model and attacker knowledge, represented by the set of knowledge K.

It is assumed that a UML subsystem S preserves the secrecy of an expression E from adversaries of type A if E does not appear in the knowledge set K of adversary A during any execution of $[[S]]_A$. In order to preserve the secrecy of a variable it is necessary to preserve the secrecy of all expressions in which this variable occurs. The secrecy of data means that only legitimate parties are allowed to read the data. Please note that sending $\{m\}_K::K \in Exp$ does not preserve secrecy, though sending only $\{m\}_K \in Exp$ does.

The integrity of data means that only legitimate parties are allowed to modify the data. Let a set $E \subseteq Exp$ represent acceptable expressions. It is assumed that a UML subsystem S preserves the integrity of an attribute a with respect to E from adversaries of type A with initial knowledge K^0 if during any execution of $[[S]]_A$, at any point, the attribute a is undefined or refers to an element of E. The $E=Exp\backslash K^0$ ("\" – subtractions of sets) means that S preserves the integrity of an attribute a with respect to E from adversaries of type A with initial knowledge K^0. In other words, the integrity of a variable means that no potential adversary is able to change the value of the variable into an unacceptable one during the execution of S.

The authenticity may consider a message or an entity. Message authenticity (i.e. data origin authenticity) means that one can trace back a piece of data to its original source. Let us assume that in a system S there exist attributes a and o, where o is supposed to store the origin of the message stored in a. S provides message authenticity of the attribute a with respect to its origin o threatened by adversaries of type A with initial knowledge K^0 if during any execution of $[[S]]_A$, at any point, the attribute a appears as the first in the sub-expression in $outQu_o$, and in all output queues and link queues in S. It means that a message has its origin in a part of the system if, during any execution of the system, the message appears in that part of the system for the first time. Entity authenticity ensures that one can identify a participant in the protocol and, in particular, make sure that the party has actually actively participated in the protocol at that time.

In the UMLsec it was assumed that the freshness of a value $data \in Data \cup Keys$ is related to its:

- unpredictability – the adversary cannot guess this value;
- newness – the value has never appeared since the time the system began to operate.

Particularly, a message is fresh if it was created during the current execution round of the considered system and therefore cannot be used by the adversary as a replay of an older message. A nonce is a good example of a fresh value.

In multi-level secure systems there are different levels of data sensitivity, e.g.: high level encompassing highly sensitive (trusted) data, and low level including less sensitive (trusted) data. The communication between these parts of the system requires a method that would prevent the sensitive data from leaking from a trusted part to an untrusted one. For that reason the UMLsec uses the following secure information flow rules:

- "no down-flow" policy, i.e. low data may influence high data and not vice versa,
- "no up-flow" policy, i.e. high data may influence low data and not vice versa.

The UMLsec stereotypes and tags are based on these property definitions. Besides, many of the mentioned above can be used. Please note that:

- stereotypes, e.g.: <<fair exchange>>, <<provable>>, <<encrypted>>, <<rbac>>, <<Internet>>, <<smart card>>, <<no up-flow>>, <<guarded>>, etc.,
- tags, e.g.: start, stop, adversary, secrecy, integrity, authenticity, role, etc.,
- constraints, e.g.: "action is non-deniable", "prevents down-flow", etc.,

are added to the subset of standard UML elements, called there "base classes", e.g.: subsystem, link, node, dependency, object – to better, i.e. more precisely, specify security properties of the modelled security-related product or system.

5. ELABORATION OF THE TOE SECURITY ENVIRONMENT

5.1. TOE security environment specification defined by the standard

The second main development stage is establishing the TOE security environment (in the CC v. 3.x called "Security problem definition"). Its purpose is to define the nature and scope of the security needs (concerns) to be addressed by the TOE [38]. The Common Criteria evaluation is generally oriented towards the products (including system products), rather than working systems. The systems are built using certified and not certified products. The standard says that this requires precise and realistic specification of the working environment, including assumptions, threats and security policy rules that must comply. This concept can be expressed as the class diagram, shown in the Fig. 5.1. It will be the basis for further developed models presented in this monograph. The ITSDF framework has many facilities to issue precise specifications of the security environment – the methodology and the specification language, especially an enhanced set of security environment generics, including OSPs, compliant with the common security management standards, e.g. [58].

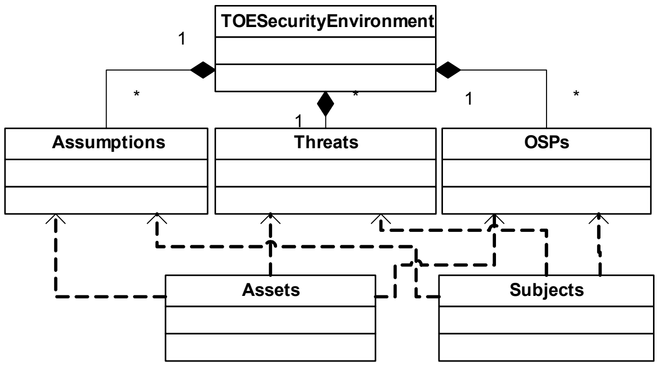


Fig. 5.1. TOE security environment – the CC defined basic elements
Rys. 5.1. Otoczenie zabezpieczeń przedmiotu oceny – elementy określone w standardzie Wspólne kryteria

5.2. TOE security environment data model

The below Fig. 5.2 is the proposed extension of the element structure presented in the Fig. 5.1. The figure specifies all details used for modelling key aspects of the TOE security environment and its elaboration on the basis of the ST introduction model (with the use of some BCL model elements). Please note the classes responsible for the elaboration process and those representing both specification elements.

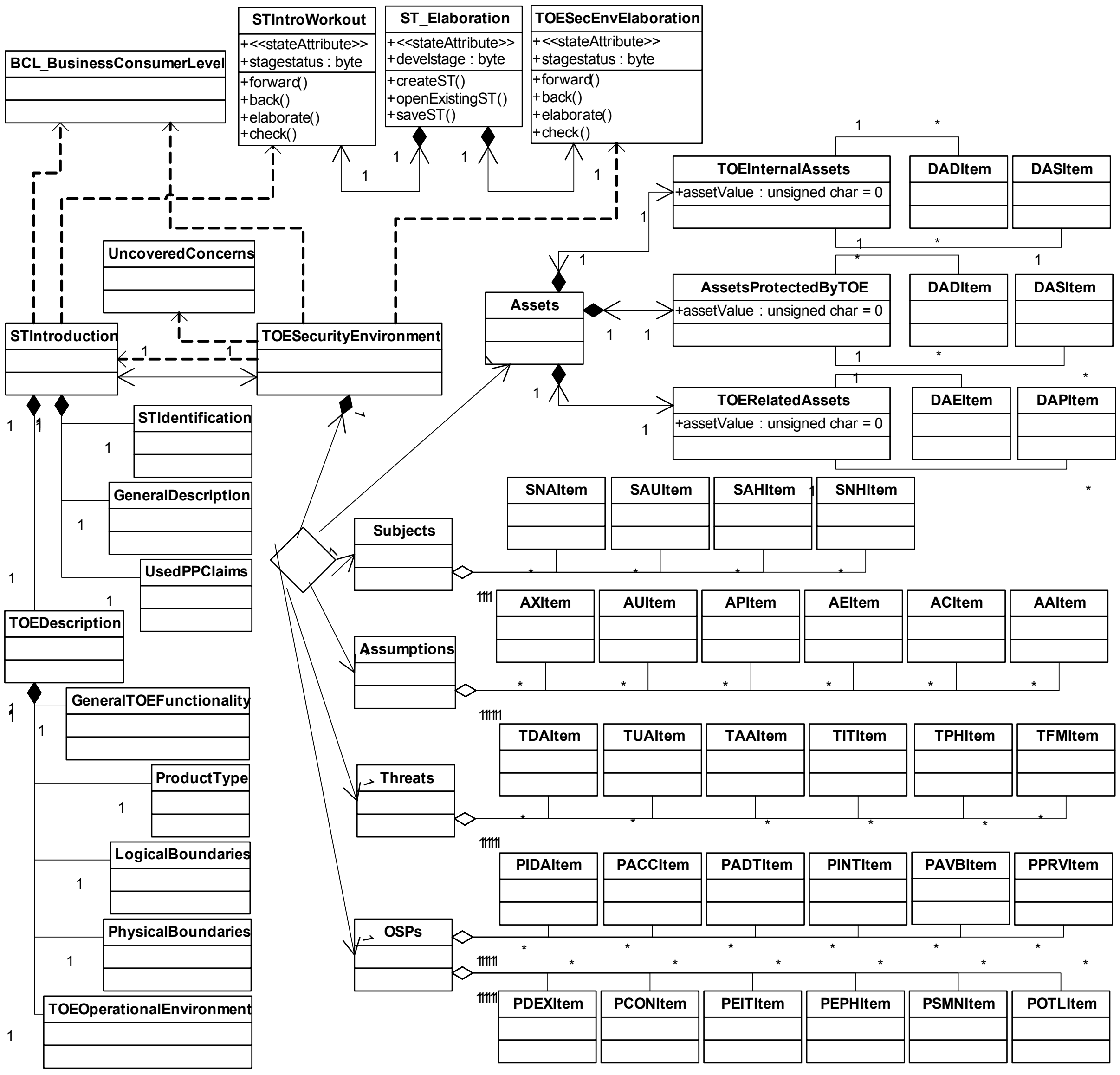


Fig. 5.2. TOE security environment elaboration on the basis of the ST introduction model – the assumed static structure diagram

Rys. 5.2. Wypracowanie specyfikacji otoczenia zabezpieczeń przedmiotu oceny na podstawie wprowadzenia do zadania zabezpieczeń – przyjęty diagram struktury statycznej

The TOE security environment is expressed by different sets grouping the generic items. Three main kinds of assets are distinguished, though all are expressed by the asset group generics. As it was mentioned earlier, generics can be predefined and placed into the library or created on demand by developers. Any generic can be refined to express adequately and more precisely unique design needs. It will be discussed later that security environment and security objectives generics have their basic inter-relations assigned. All these features facilitate the IT security development process. The key issue for the model presented there is the assumed TOE security environment paradigm, concerning risk assessment, security policy rules and assumptions features (Fig. 5.3). Please note basic triplets, containing a threat, threat source, also called threat agent, (`SgrGeneric`), threatened assets (`DAgrGeneric`). The threat scenario is considered along with the exploited vulnerability. Please note that assets have their owners (`SgrGeneric`) for management purposes, and values used during risk assessment. For every threat simple risk information was added, containing event likelihood, impacts and the operation to estimate the risk value. This simple method allows to order the risk scenarios by the risk value and helps to select proper measures.

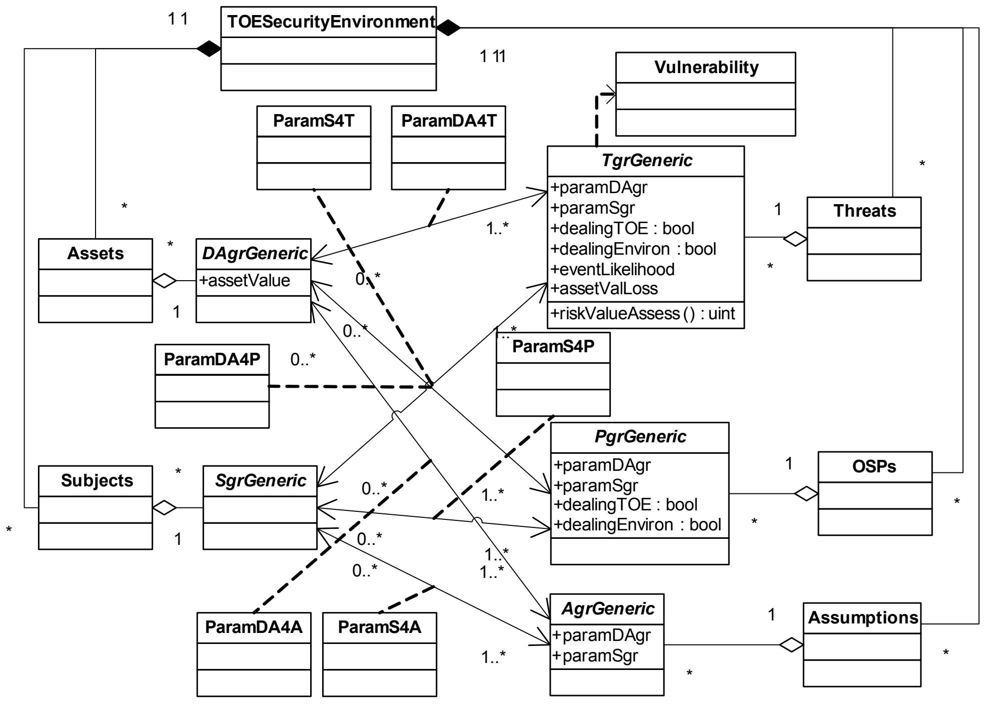


Fig. 5.3. TOE security environment paradigm
Rys. 5.3. Paradygmat otoczenia zabezpieczeń przedmiotu oceny (TOE)

The model also allows to separate items dealing with the TOE and/or its operational environment and set relations between them and the assets and subjects. Please note that it is very difficult to specify all security items at the beginning of the development process, especially to decide what should be covered by the TOE only, by its environment or by both. For this reason the presented framework allows to postpone these design decisions when more details have been known. A similar problem occurs while choosing the way of the TOE security environment specification: the threat based or the OSPs based. For this stage it is very important to provide the trade-off between its elements, especially between threats and policies, albeit generally the threat specification has higher priority.

5.3. TOE security environment elaboration process

The general elaboration scheme (the behaviour of the `TOESecEnvElaboration` class), as the UML activity diagram, was shown in the Fig. 5.4.

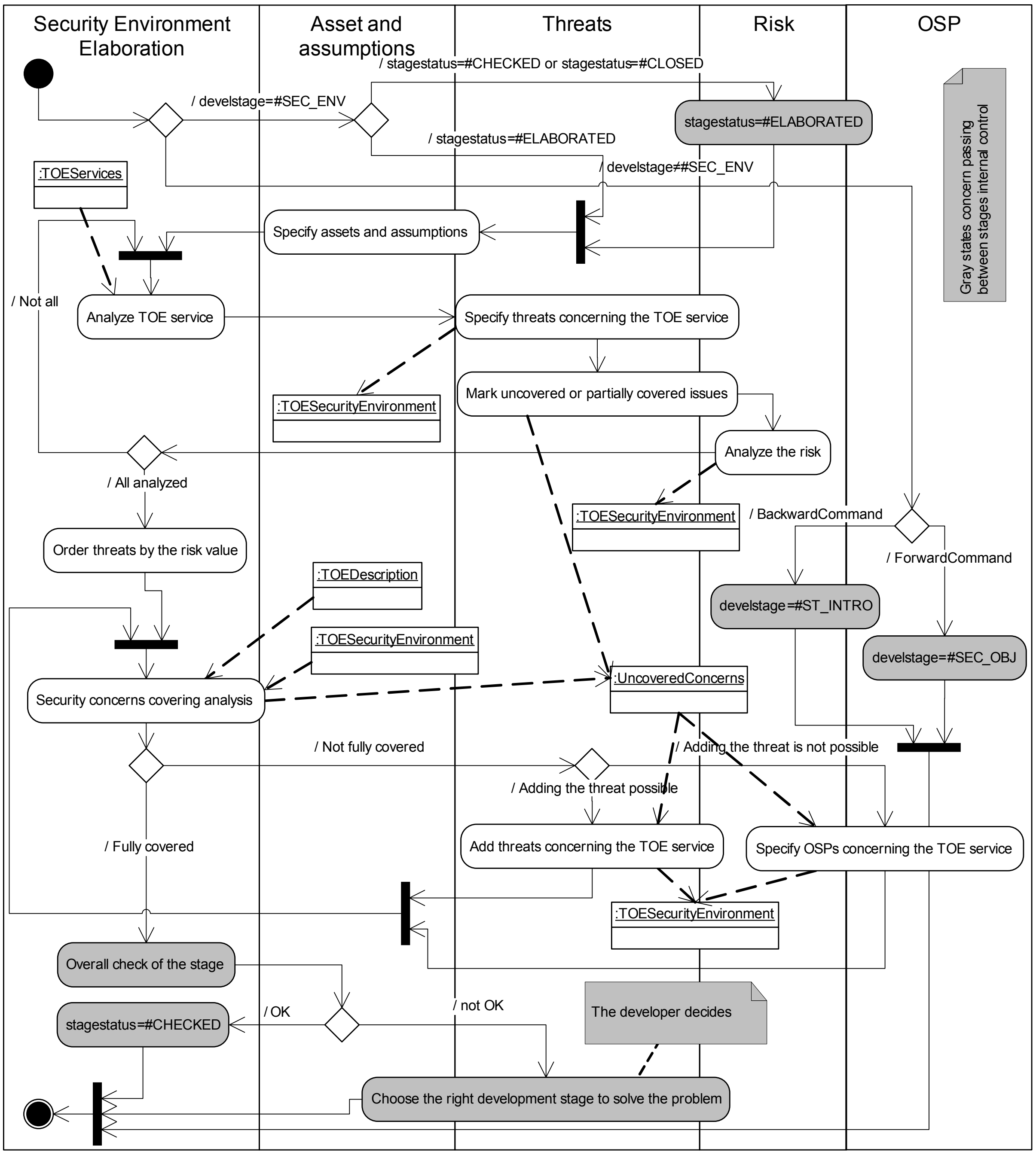

Security Environment Elaboration
Asset and assumptions
Threats
Risk
OSP
/ stagestatus=#CHECKED or stagestatus=#CLOSED
/ develstage=#SEC_ENV
stagestatus=#ELABORATED
/ stagestatus=#ELABORATED
/ develstage≠#SEC_ENV
Gray states concern passing between stages internal control
:TOEServices
Specify assets and assumptions
/ Not all
Analyze TOE service
Specify threats concerning the TOE service
Mark uncovered or partially covered issues
:TOESecurityEnvironment
Analyze the risk
/ All analyzed
:TOESecurityEnvironment
/ BackwardCommand
/ ForwardCommand
Order threats by the risk value
:TOEDescription
develstage=#ST_INTRO
develstage=#SEC_OBJ
:TOESecurityEnvironment
Security concerns covering analysis
:UncoveredConcerns
/ Not fully covered
/ Adding the threat is not possible
/ Adding the threat possible
/ Fully covered
Add threats concerning the TOE service
Specify OSPs concerning the TOE service
:TOESecurityEnvironment
Overall check of the stage
/ OK
/ not OK
The developer decides
stagestatus=#CHECKED
Choose the right development stage to solve the problem

Fig. 5.4. TOE security environment elaboration – the general activity diagram
Rys. 5.4. Wypracowanie specyfikacji otoczenia zabezpieczeń – nadrzędny diagram czynności

The diagram presents main and auxiliary swimlanes which will be considered below on separate diagrams. Please note that this TOE security environment elaboration diagram is “risk-based” and represents “threat approach”, while only those items that could not be expressed by threats are expressed by OSPs. This is generally preferable as a more concise and precise means.

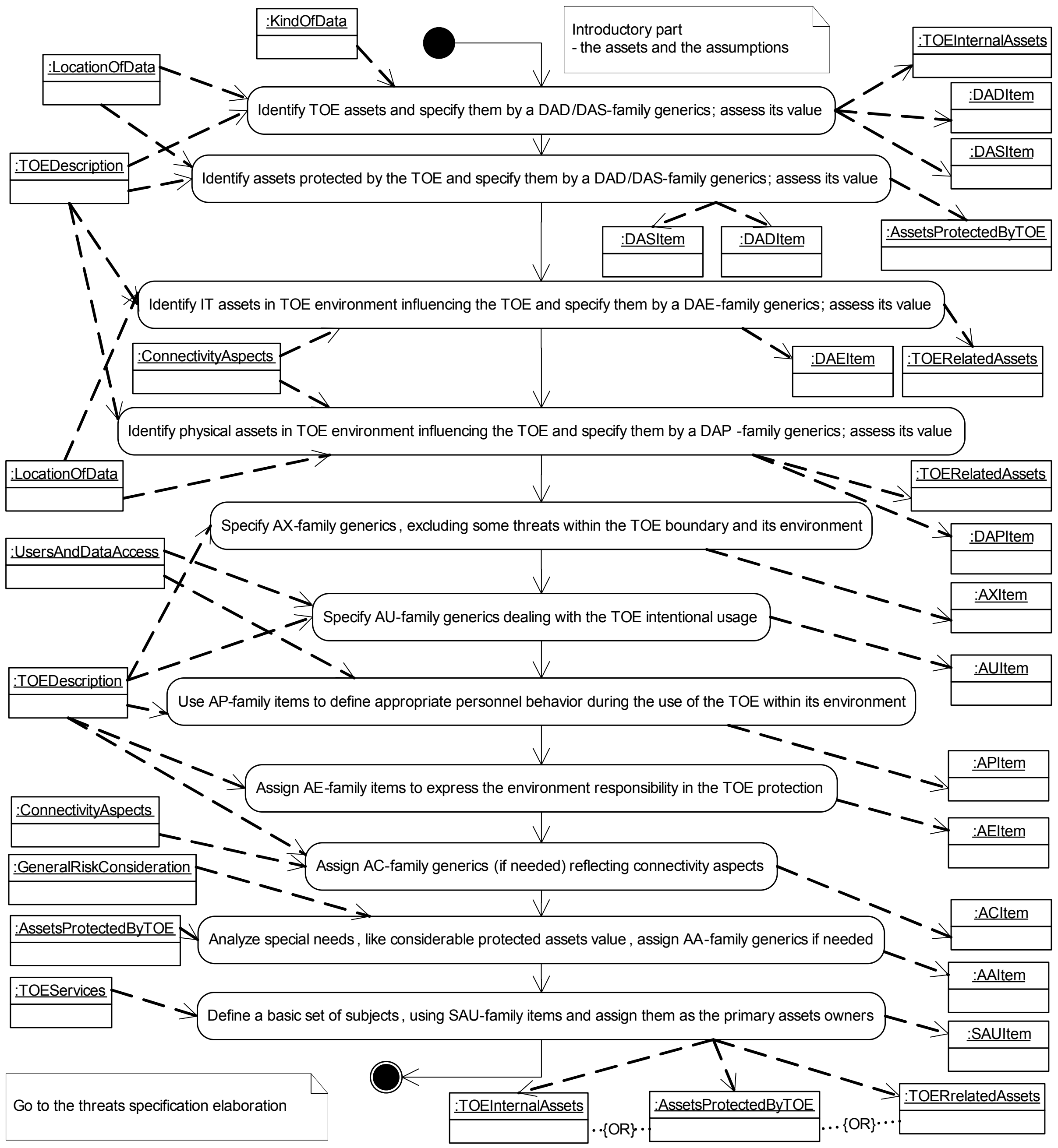

:KindOfData
Introductory part
- the assets and the assumptions
:TOEInternalAssets
:LocationOfData
Identify TOE assets and specify them by a DAD/DAS-family generics; assess its value
:DADItem
:DASItem
:TOEDescription
Identify assets protected by the TOE and specify them by a DAD/DAS-family generics; assess its value
:DASItem
:DADItem
:AssetsProtectedByTOE
Identify IT assets in TOE environment influencing the TOE and specify them by a DAE-family generics; assess its value
:ConnectivityAspects
:DAEItem
:TOERelatedAssets
Identify physical assets in TOE environment influencing the TOE and specify them by a DAP -family generics; assess its value
:LocationOfData
:TOERelatedAssets
Specify AX-family generics, excluding some threats within the TOE boundary and its environment
:DAPItem
:UsersAndDataAccess
:AXItem
Specify AU-family generics dealing with the TOE intentional usage
:AUItem
:TOEDescription
Use AP-family items to define appropriate personnel behavior during the use of the TOE within its environment
:APItem
Assign AE-family items to express the environment responsibility in the TOE protection
:ConnectivityAspects
:AEItem
Assign AC-family generics (if needed) reflecting connectivity aspects
:GeneralRiskConsideration
:ACItem
:AssetsProtectedByTOE
Analyze special needs, like considerable protected assets value, assign AA-family generics if needed
:AAItem
:TOEServices
Define a basic set of subjects, using SAU-family items and assign them as the primary assets owners
:SAUItem
Go to the threats specification elaboration
:TOEInternalAssets
{OR}
:AssetsProtectedByTOE
{OR}
:TOERrelatedAssets

Fig. 5.5. TOE security environment elaboration – the introductory part diagram (assets and assumptions)
Rys. 5.5. Wypracowanie specyfikacji otoczenia zabezpieczeń – część wstępna (zasoby i założenia na otoczenie)

The first subordinated diagram (Fig. 5.5) presents introductory TOE developer's activities, attempted at specifying assets and assumptions. Three different assets specifications were assumed: included within the TOE, protected by the TOE, and those auxiliary environmental assets which should be protected for the whole TOE security.

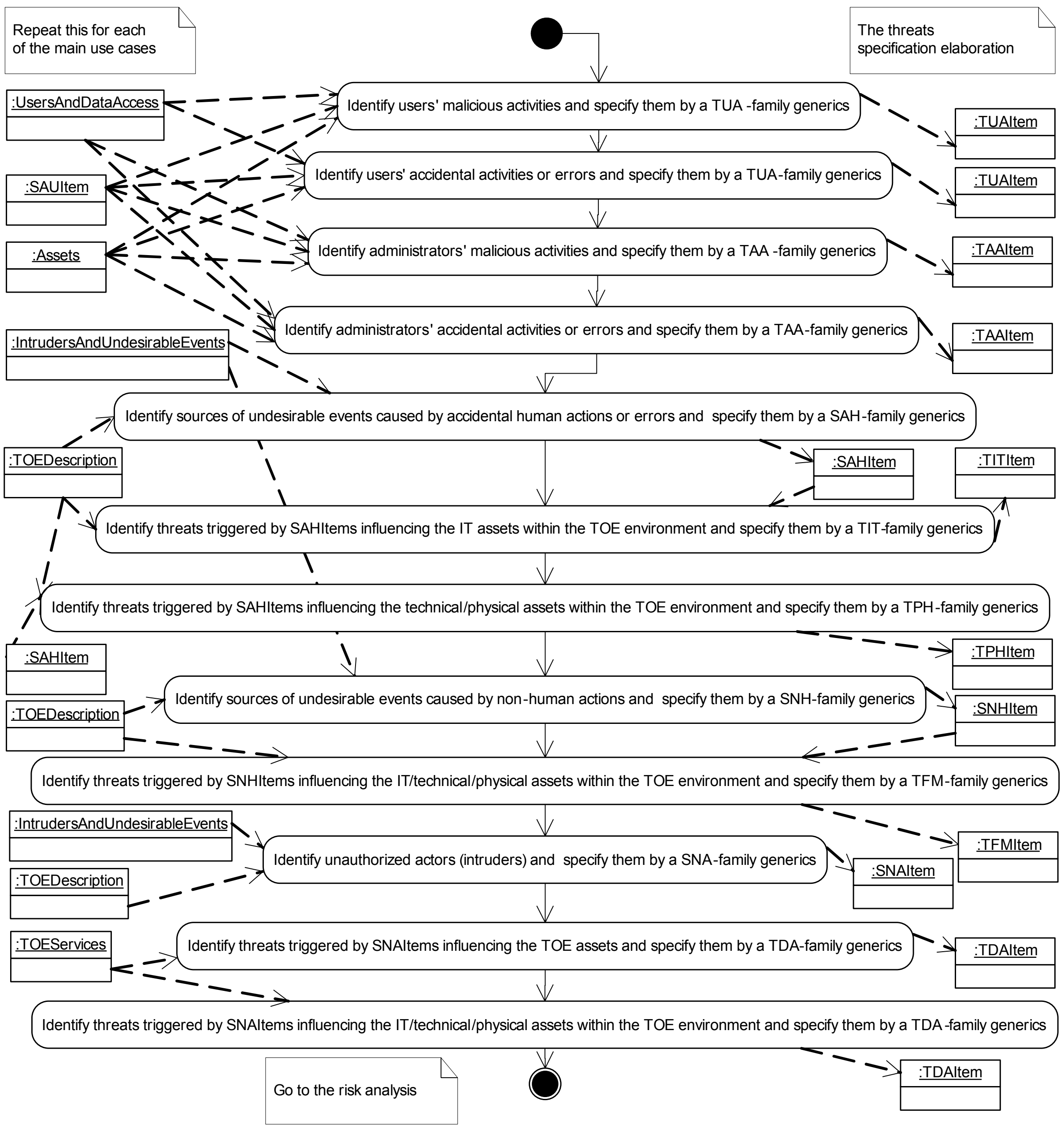

Repeat this for each of the main use cases
The threats specification elaboration
:UsersAndDataAccess
Identify users' malicious activities and specify them by a TUA -family generics
:TUAItem
:SAUItem
Identify users' accidental activities or errors and specify them by a TUA-family generics
:TUAItem
:Assets
Identify administrators' malicious activities and specify them by a TAA -family generics
:TAAItem
:IntrudersAndUndesirableEvents
Identify administrators' accidental activities or errors and specify them by a TAA-family generics
:TAAItem
Identify sources of undesirable events caused by accidental human actions or errors and specify them by a SAH-family generics
:TOEDescription
:SAHItem
:TITItem
Identify threats triggered by SAHItems influencing the IT assets within the TOE environment and specify them by a TIT-family generics
Identify threats triggered by SAHItems influencing the technical/physical assets within the TOE environment and specify them by a TPH-family generics
:SAHItem
:TPHItem
:TOEDescription
Identify sources of undesirable events caused by non-human actions and specify them by a SNH-family generics
:SNHItem
Identify threats triggered by SNHItems influencing the IT/technical/physical assets within the TOE environment and specify them by a TFM-family generics
:IntrudersAndUndesirableEvents
:TOEDescription
Identify unauthorized actors (intruders) and specify them by a SNA-family generics
:SNAItem
:TFMItem
:TOEServices
Identify threats triggered by SNAItems influencing the TOE assets and specify them by a TDA-family generics
:TDAItem
Identify threats triggered by SNAItems influencing the IT/technical/physical assets within the TOE environment and specify them by a TDA -family generics
:TDAItem
Go to the risk analysis

Fig. 5.6. TOE security environment elaboration – the threat specification diagram
Rys. 5.6. Wypracowanie specyfikacji otoczenia zabezpieczeń – specyfikowanie zagrożeń

The developed STs are considered more useful when written for a specific environment and when they take into account risk assessment results [97]. For this reason a simple risk analyzer was built into the IT Security Development Framework. The second subordinated diagram (Fig. 5.6) presents threat specifications of different families. During these activities some other actors and their generics are identified – unauthorized actors and different sources of undesirable events.

The analyzed assets requiring protection (`DAgrGeneric`), their value and vulnerabilities, potential threats agents, sources of undesirable events, etc., allow to create a list of possible threats dealing with the TOE and/or its environment. By adding the assessed threats likelihood and impacts (percentage of the asset value loss), the risk value for these threats is assessed (Fig. 5.7). It allows to order threats by the risk value.

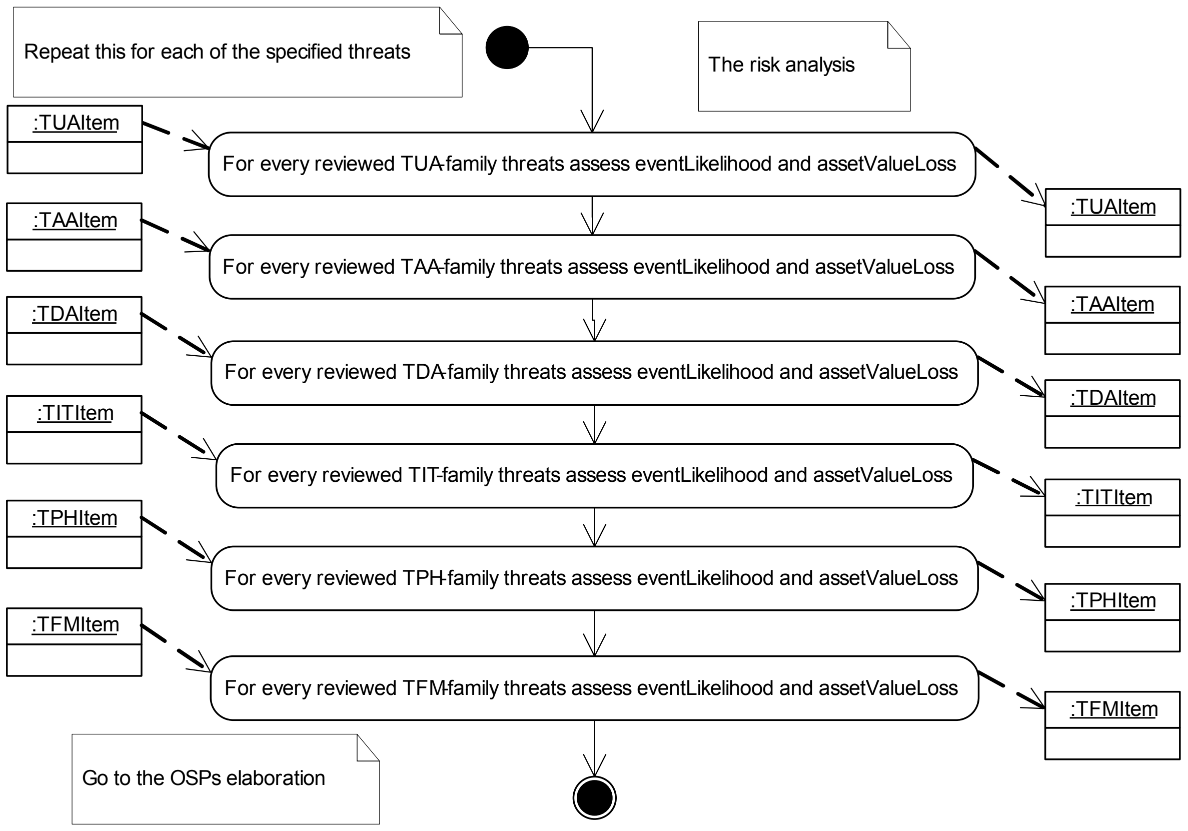

Repeat this for each of the specified threats
The risk analysis
:TUAItem
For every reviewed TUA-family threats assess eventLikelihood and assetValueLoss
:TAAItem
For every reviewed TAA-family threats assess eventLikelihood and assetValueLoss
:TDAItem
For every reviewed TDA-family threats assess eventLikelihood and assetValueLoss
:TITItem
For every reviewed TIT-family threats assess eventLikelihood and assetValueLoss
:TPHItem
For every reviewed TPH-family threats assess eventLikelihood and assetValueLoss
:TFMItem
For every reviewed TFM-family threats assess eventLikelihood and assetValueLoss
Go to the OSPs elaboration

Fig. 5.7. TOE security environment elaboration – the risk assessment diagram
Rys. 5.7. Wypracowanie specyfikacji otoczenia zabezpieczeń – ocena ryzyka

The last swimlane of the Fig. 5.4 encompasses the elaboration of the OSP specification (Fig. 5.8). It was assumed the lowest priority in comparison with the threats specification, as it is suggested in [60]. For every family of the OSPs, the three basic elements are most important: the assets – internal, externally protected by the TOE or related, expressed by the DAD family/DAS family generics, the authorized subjects (SAU family) and the given policy rule, usually strongly related to the TOE service. The Fig. 5.8 presents key input objects considered during the OSPs elaboration.

It is important to review specified threats against the security concerns, revealing topics not covered, not fully covered, or difficult to cover by threats, and cover them by defining OSPs. Usually some assumptions described by the AU-family or AP-family generics should be modified too.

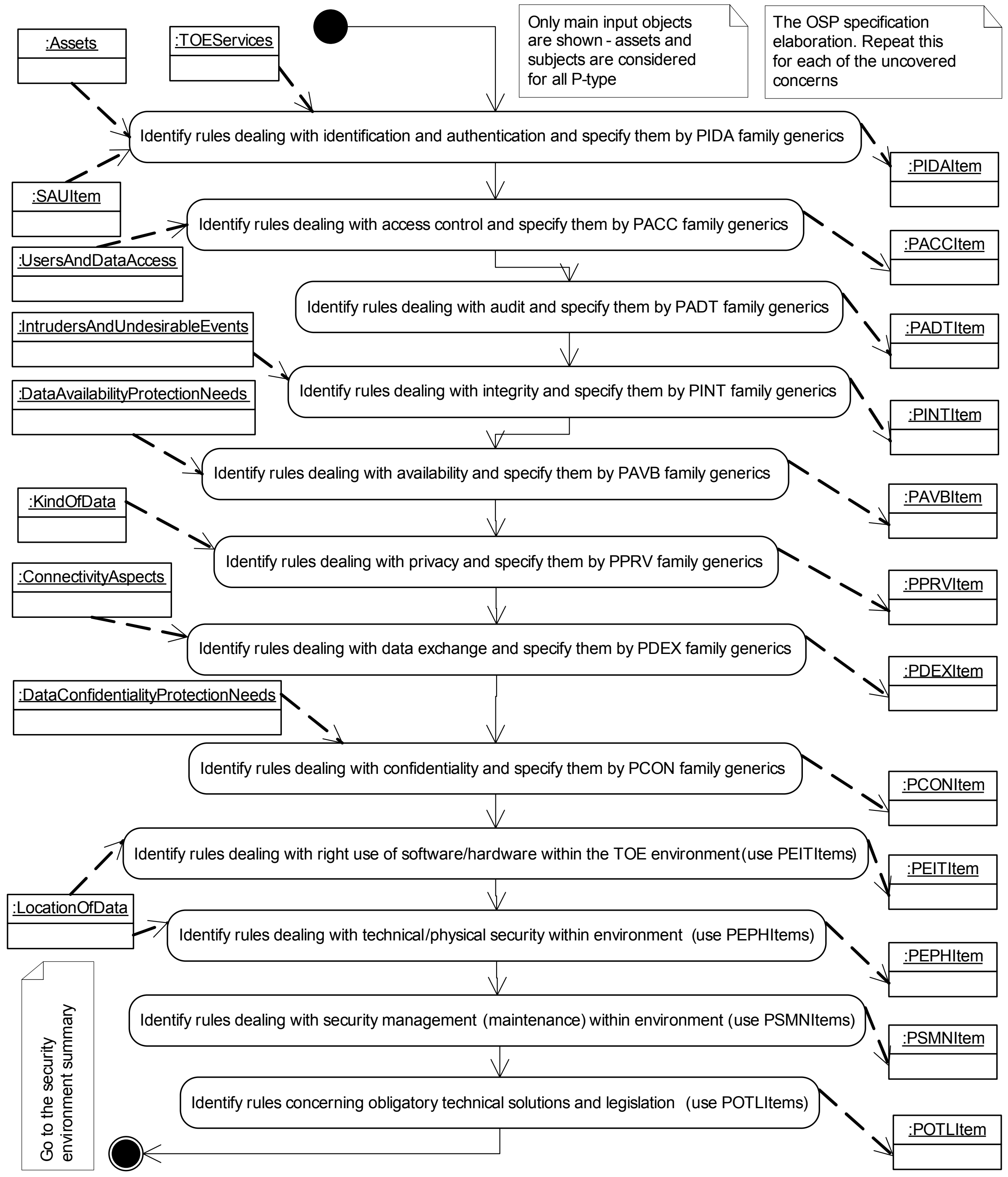

Only main input objects are shown - assets and subjects are considered for all P-type
The OSP specification elaboration. Repeat this for each of the uncovered concerns
:Assets
:TOEServices
:SAUItem
:UsersAndDataAccess
:IntrudersAndUndesirableEvents
:DataAvailabilityProtectionNeeds
:KindOfData
:ConnectivityAspects
:DataConfidentialityProtectionNeeds
:LocationOfData
Identify rules dealing with identification and authentication and specify them by PIDA family generics
Identify rules dealing with access control and specify them by PACC family generics
Identify rules dealing with audit and specify them by PADT family generics
Identify rules dealing with integrity and specify them by PINT family generics
Identify rules dealing with availability and specify them by PAVB family generics
Identify rules dealing with privacy and specify them by PPRV family generics
Identify rules dealing with data exchange and specify them by PDEX family generics
Identify rules dealing with confidentiality and specify them by PCON family generics
Identify rules dealing with right use of software/hardware within the TOE environment (use PEITItems)
Identify rules dealing with technical/physical security within environment (use PEPHItems)
Identify rules dealing with security management (maintenance) within environment (use PSMNItems)
Identify rules concerning obligatory technical solutions and legislation (use POTLItems)
:PIDAItem
:PACCItem
:PADTItem
:PINTItem
:PAVBItem
:PPRVItem
:PDEXItem
:PCONItem
:PEITItem
:PEPHItem
:PSMNItem
:POTLItem
Go to the security environment summary

Fig. 5.8. TOE security environment elaboration – the OSP specification diagram
Rys. 5.8. Wypracowanie specyfikacji otoczenia zabezpieczeń – reguły polityki bezpieczeństwa

The OSPs identification finishes the TOE security environment specification, though this is rather a recursive process and some other items can be added, removed or modified later.

5.4. Use cases for threat scenarios analysis

In developing the TOE security environment it is very helpful to analyze use case diagrams.

They express TOE functionality, i.e. the expected behaviour, and also support the TOE environment modelling with threats that may disturb this behaviour.

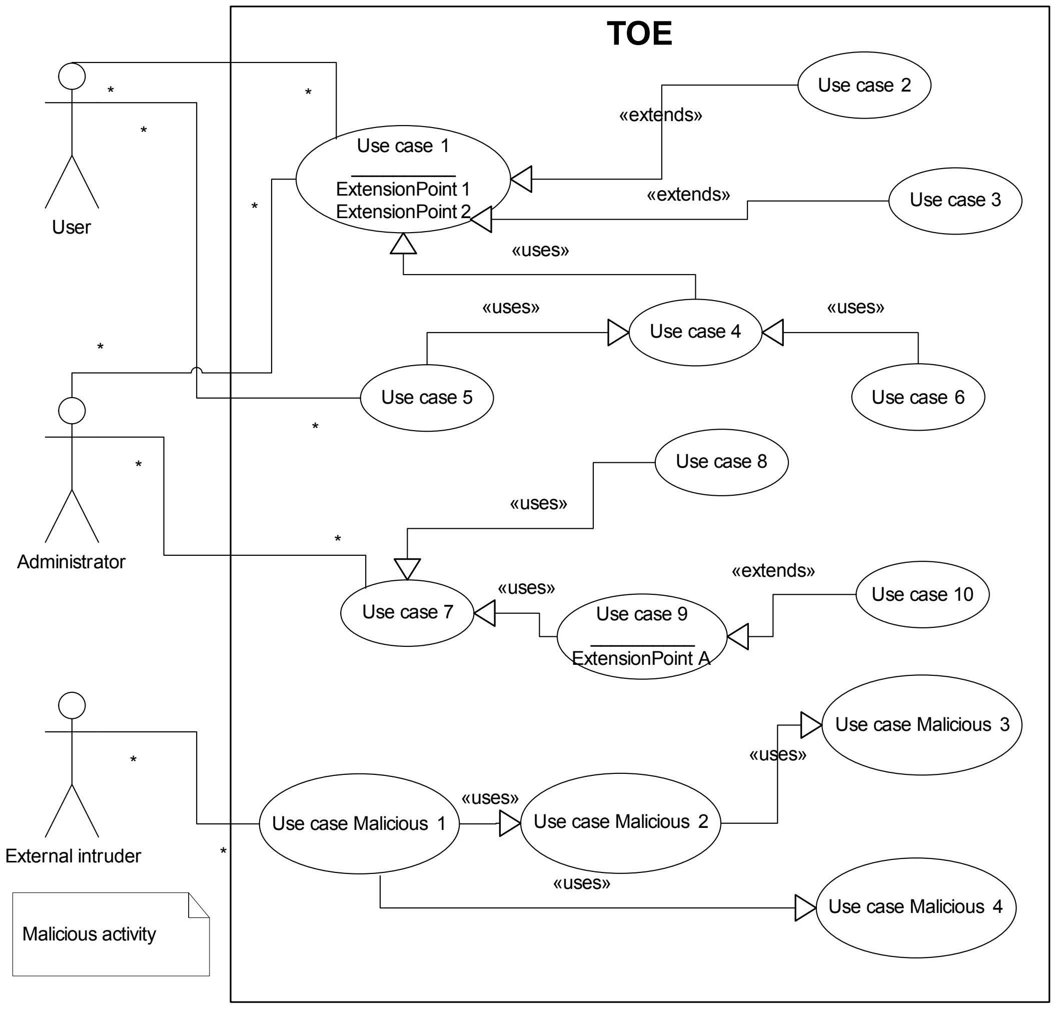


Fig. 5.9. Use case diagram with legal and illegal interacting actors
Rys. 5.9. Diagram przypadków użycia z uprawnionymi i nieuprawnionymi aktorami prowadzącymi interakcje

In the real world there are many use cases and actors, legal or illegal, assigned to them. It is convenient to represent them all on a use case diagram (Fig. 5.9) which shows actors, use cases and their associations. In comparison with the Fig. 4.9, an additional use case dealing with the intruder's malicious activity was added. It should be noted that use cases may be very convenient to identify risk scenarios, especially with malicious cascading effects.

5.5. Selected UMLsec issues concerning the TOE security environment

The UMLsec methodology [72] has introduced specific formal issues that ought to be interpreted with the use of the Common Criteria methodology. All information concerning assets, assumptions, security policies, and first and foremost threats should be properly extracted from the UMLsec models during the TOE security environment elaboration, though the UMLsec does not directly provide security issues compatible with the Common Criteria methodology.

The UMLsec allows to model potential adversary behaviour. The set of abstract threats is introduced: $\{delete, read, insert, access\}$, emerging from a physical layer of the system. The $access$ (an adversary may access to the physical node) concerning the system node can be decomposed to atomic actions: $delete$, $read$ and $insert$, with respect to the communication links connected with this node.

The $Threats_A(s)$ function specifies the threat scenario related to the adversary type A against a component or link expressed by the stereotype s, and returns a subset of $\{delete, read, insert, access\}$. From these abstract threats concrete threats are derived, used for modelling a specific behaviour, e.g.: for a link or node x in a deployment diagram in a UML subsystem specification S, it can be expressed as: $threats^S_A(x)$. The security evaluation of the system is performed with respect to a given type of adversary and the knowledge he/she possesses.

Specifying the TOE security environment, the security properties of the IT product or system, specified in the UMLsec, ought to be expressed by the *TgrGeneric*, *AgrGeneric*, and *PgrGeneric* generics.

For each case of $Threats_A(s)$ the right *TgrGeneric* generics can be identified within the library or defined. Please note that:

- $Threats$ related to the threat scenario with returned value, i.e. $delete$, $read$, $insert$, $access$, can be expressed by the generic *mnemonic*, *description* and *refinement* fields,
- A, representing an adversary, can be expressed by *paramSgr*, substituted or not by the attacker *SgrGeneric*,
- s, representing a component or link being the threatened asset, can be expressed by *paramDAgr*, substituted or not by the *DAgrGeneric* asset.

The UMLsec stereotypes with related tags presenting the security properties can be expressed with the assumptions (*AgrGeneric*) or OSPs (*PgrGeneric*), e.g.:

- <<rbac>> is directly related to the existing *PACC.RBAC* policy generic,
- <<fair exchange>> can be the basis for the *PDEX.FairExchange* generic definition for e-business applications,

- <<no down-flow>> is de facto information flow policy, not available in the current version of the generic library, but can be defined by developers, as *PDEX.NoDownFlow,*
- <<Internet>>, <<LAN>> can be the basis to define *AC.Internet*, *AC.LAN*, while <<encrypted>> can be the basis for *PCON.EncData* generics definitions.

The TOE security environment stage plays the key role in transforming the IT product or system (TOE) model to its CC-compliant IT security development model (i.e. ST or PP), regardless of the methodology applied to create the IT product or system model. These issues concern the ADV assurance class. All security relevant properties of the TOE ought to be extracted from the TOE model. It will be shown on the selected issues concerning the UMLsec as the TOE model specification methodology. Please refer also to the example 3.5 which deals with an informal TOE representation.

Example 5.1: UMLsec TOE model interpretation during TOE security environment elaboration – selected issues.

The Fig. 5.10 shows a part of the distributed data communication system, called there Secure channel, based on cryptography (similar to broadly implemented SSL – Secure Socket Layer protocol). Please note the Client (e.g. a web browser) and Server (e.g. a web server) classes and the objects corresponding to them and exchanging information according to the assumed protocol.
Please note UMLsec stereotypes:

- <<data security>> of the Secure channel system meaning that the [72] "system provides secrecy, integrity, authenticity, freshness" as a whole with respect to the given adversary model;
- both the Client and Server classes are <<critical>> and this criticality is expressed in more detail using the corresponding tags, i.e. secrecy, integrity, authenticity, freshness.

Please note protocol data and operations declared for both classes. The data exchange is initiated from the client side. The web browser sends (invoking `init()` operation) the following to the server: the nonce N, the client public key K_C, and the record containing the client name and client public key K_C signed with the use of its private key K_C^{-1}:

`init(`$N, K_C, Sign_{K_C^{-1}}(C::K_C)$`).`

The server response to this invoking:

`resp(`$\{Sign_{K_S^{-1}}(K::N::K_C)\}_{K_C}, Sign_{K_{CA}^{-1}}(S::K_S)$`),`

where the meaning of the arguments is:

$\{Sign_{K_S^{-1}}(K::N::K_C)\}_{K_C}$ – the encrypted record, using the client public key K_C and including the symmetric server generated session key K, returned the N nonce value and the client public key K_C authenticating the server,

$Sign_{K_{CA}^{-1}}(S::K_S)$ – the digital certificate of the server public key K_S.

Using a common session, the key K client can send the confidential data s invoking:

`xchd(`$\{s\}_K$`).`

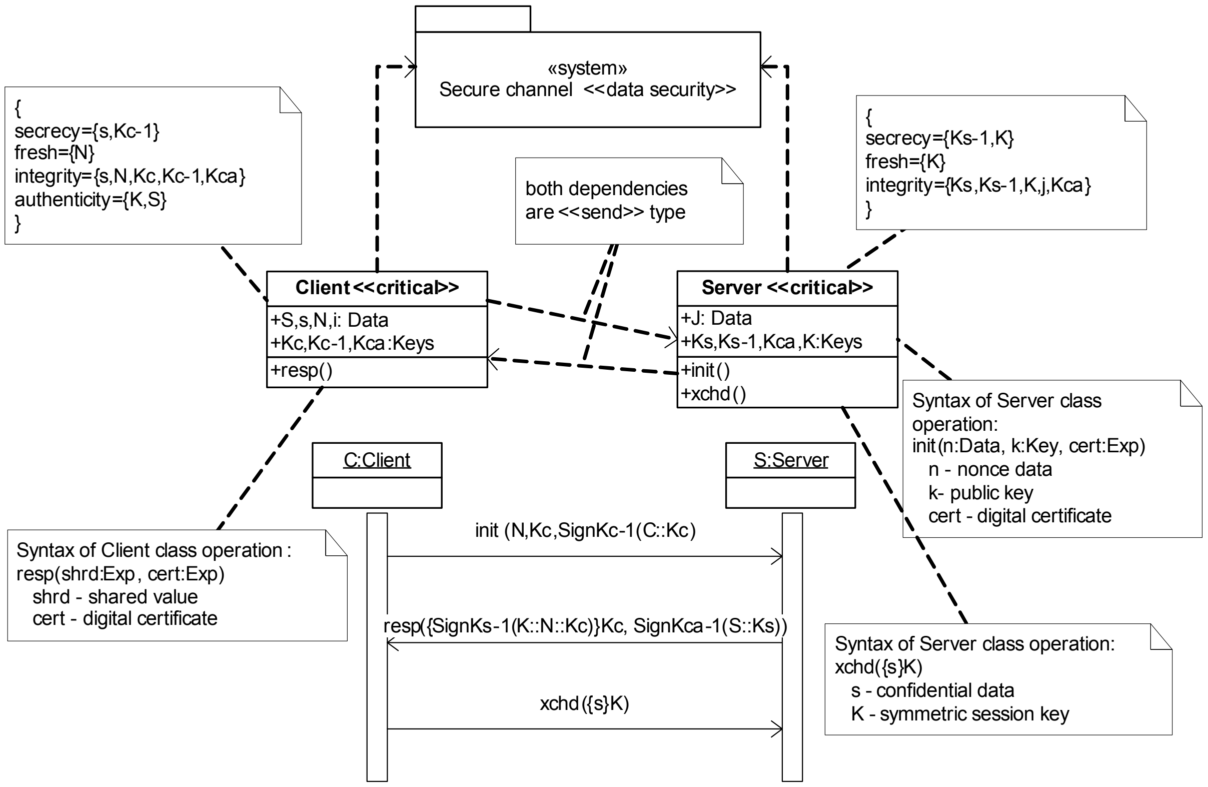

«system»
Secure channel <<data security>>
{
secrecy={s,Kc-1}
fresh={N}
integrity={s,N,Kc,Kc-1,Kca}
authenticity={K,S}
}
both dependencies are <<send>> type
{
secrecy={Ks-1,K}
fresh={K}
integrity={Ks,Ks-1,K,j,Kca}
}
Client <<critical>>
+S,s,N,i: Data
+Kc,Kc-1,Kca:Keys
+resp()
Server <<critical>>
+J: Data
+Ks,Ks-1,Kca,K:Keys
+init()
+xchd()
Syntax of Server class operation:
init(n:Data, k:Key, cert:Exp)
n - nonce data
k- public key
cert - digital certificate
C:Client
S:Server
init (N,Kc,SignKc-1(C::Kc)
resp({SignKs-1(K::N::Kc)}Kc, SignKca-1(S::Ks))
xchd({s}K)
Syntax of Client class operation :
resp(shrd:Exp, cert:Exp)
shrd - shared value
cert - digital certificate
Syntax of Server class operation:
xchd({s}K)
s - confidential data
K - symmetric session key

Fig. 5.10. Key exchange protocol implementation – an example
Rys. 5.10. Implementacja protokołu wymiany klucza – przykład

Analyzing this simplified protocol implemented within the TOE, the developer ought to specify the TOE security environment elements, like assumptions, threats and OSP policies. The <<data security>> stereotype applied for the security channel requires to consider a given adversary model. For this reason, the developers use and refine, for example, the following generic related to the man-in-the-middle type attack:

TDA.ModInfor_D0I0. Message content modification. A hacker [paramSNA] modifies information intercepted from a communication link between two unsuspecting entities before passing it on, thereby deceiving the intended recipient.
Refinement: This manipulation is to exploit any design flaws that may exist in the cryptographic protocol.

The developer also assumes that:
AX.ResistCrypto_D0I0. Adversary is not able to break the encryption system implemented in the protocol (a user-defined generic).
AE.StronKey_D1I0. Application [paramDAS] uses only appropriate secret keys (chosen from a large key space) as input for the cryptographic function of the TOE to ensure the strength of cryptographic operation (a derived generic).

The developer can add other generics and continue the development until successful rationale process.

□

This example shows that the TOE development can base on the UMLsec formal methodology and the TOE security development can base on the methodology presented in the monograph. Finally, they can meet together where security-related IT product or system is specified according to the ADV assurance class requirements.

5.6. Formal approach to the TOE security environment specification

The Common Criteria IT security development methodology allows less or more formal approaches. A reasonable level of formalization can bring many advantages (the development time, cost, preciseness, etc.). The level of the model formalization depends on the user's need and her/his ability to implement the selected formal approach. The methodology presented in the monograph has a semiformal character. It can be complemented by some formal elements to create more precise models.

Definition 5.1: TOE security environment.

The TOE security environment is a pair of sets:

$S_{TOESecEnv} = \langle S_{TOESecEnvItems}, S_{TOESecEnvPars} \rangle$.

1. The first element of the pair, $S_{TOESecEnvItems}$ is the sum of five disjointed sets of generics:

$S_{TOESecEnvItems} = S_{Subjects} \cup S_{Assets} \cup S_{Threats} \cup S_{OSPs} \cup S_{Assumpt}$ and

$S_{Subjects} \cap S_{Assets} \cap S_{Threats} \cap S_{OSPs} \cap S_{Assumpt} = \varnothing$,

where:

$S_{Subjects} = \{\texttt{SgrGeneric} \mid \texttt{SgrGeneric}.assignstat{=}\#\text{ASSIGNED}\}$,

$S_{Assets} = \{\texttt{DAgrGeneric} \mid \texttt{DAgrGeneric}.assignstat{=}\#\text{ASSIGNED}\}$,

$S_{Threats} = \{\texttt{TgrGeneric} \mid \texttt{TgrGeneric}.assignstat{=}\#\text{ASSIGNED}\}$,

$S_{OSPs} = \{\texttt{PgrGeneric} \mid \texttt{PgrGeneric}.assignstat{=}\#\text{ASSIGNED}\}$,

$S_{Assumpt} = \{\texttt{AgrGeneric} \mid \texttt{AgrGeneric}.assignstat{=}\#\text{ASSIGNED}\}$.

2. The second element of the pair, $S_{TOESecEnvPars}$ is the sum of six sets of parameterization associations:

$S_{TOESecEnvPars} \subseteq \text{GenParAssoc}$ and $S_{TOESecEnvPars} =$

$\{\texttt{ParamDA4T}, \texttt{ParamS4T}, \texttt{ParamDA4P}, \texttt{ParamS4P}, \texttt{ParamDA4A}, \texttt{ParamS4A}\}$.

□

This definition is exemplified by two figures: Fig. 5.2 and Fig. 5.3. The parameters should be properly assigned or left empty. Assuming that the types of parameter attributes are: *paramDAgr:*`DAgrGeneric` and *paramSgr:*`SgrGeneric`, this condition can be easily expressed as the OCL invariants:

```
ParamDA4T
TgrGeneric.paramDAgr=DAgrGeneric or DAgrGeneric->isEmpty
ParamDA4P
PgrGeneric.paramDAgr=DAgrGeneric or DAgrGeneric->isEmpty
ParamDA4A
AgrGeneric.paramDAgr=DAgrGeneric or DAgrGeneric->isEmpty
ParamS4T
TgrGeneric.paramSgr=SgrGeneric or SgrGeneric->isEmpty
ParamS4P
PgrGeneric.paramSgr=SgrGeneric or SgrGeneric->isEmpty
ParamS4A
```

```
AgrGeneric.paramSgr=SgrGeneric or SgrGeneric->isEmpty
□
```

For the risk value assessment operation the OCL constraints can be defined as follows (Fig. 5.3):

```
TgrGeneric::riskValueAssess()
-- check if declared asset exists
pre: self.paramDAgr->notEmpty
-- check if declared value is in the assumed range:
    and (self.paramDAgr.assetValue>=MIN_ASSET_VALUE and
    self.paramDAgr.assetValue<=MAX_ASSET_VALUE)
    and (self.assetValLoss>=MIN_ASSET_VAL_LOSS and
    self.assetValLoss <=MAX_ASSET_VAL_LOSS)
    and (self.eventLikelihood >=MIN_EV_LIKELIHOOD and
    self.eventLikelihood <=MAX_EV_LIKELIHOOD)
post:result= self.paramDAgr.assetValue*self.assetValLoss*self.eventLikelihood
□
```

The definition is on the generic group level, though the risk is estimated on the generic items level, where there exists a concrete threat, threatened asset and intruder or undesirable event represented by generic items (`GenItem`).

The security environment encompasses disjointed sets of different types of generic items (including their instances). The OCL offers different operations on collections that can be very helpful in defining quantity measures for the model, the model analysis and its optimization. These features are important in the design complexity analysis that will be shown in the selected examples. The operations presented below can be used for other stages as well.

Example 5.2: Counting the generic items of a given type placed in the specification using the OCL expression.

Let us assume for example that the Threats class has the `noOfTDAItems:Integer` attribute defined (not shown on the class diagrams) to store the number of generics of the given type (including instances):

```
Threats
self.noOfTDAItems=self.tDAItem[14]->size
□
```

Example 5.3: Checking the uniqueness of the specification items names.

Please note that all items within the specification should be unique with respect to their *mnemonic* name, derivation number and instance number. This property can be expressed as the OCL constraints, e.g. (see Fig. 3.22):

```
Threats
self.tDAItem->count(crpAnal_D0I0)=0
    -- when the given generic item was not placed into the specification
or self.tDAItem->count(crpAnal_D0I0)=1
    -- otherwise
--
-- or in a more compact way:
self.tDAItem->includes(crpAnal_D0I0)
```

[14] According to the OCL convention, the class names used in the navigations are written using the first lower letter.

```
-- returns true if this item was placed, false otherwise
□
```

Example 5.4: Finding the selected specification items using the OCL `select` operation.

Let us assume that the developer analyzes OSP rules concerning availability.

```
OSPs
self.pAVBItem
-- returns the set of all PAVBItem items placed in the specification
self.pAVBItem->size
-- count them
self.pAVBItem->select(dealingTOE=true and dealingEnviron=false)
-- only the rules concerning TOE are selected
□
```

Example 5.5: Creating the complete specifications.

Let `allassumptions:Set` be the set containing all items of the assumptions specification. It is the Assumptions class attribute.
Please note that `Assumptions.allassumptions` = $S_{Assumpt}$, and, in a similar way, other TOE security environment subsets can be extracted from the model.

```
Assumptions
allassumptions=self.aXItem->union
                 (self.aUItem->union
                 (self.aPItem->union
                 (self.aEItem->union
                 (self.aCItem->union
                 (self.aAItem)))))
□
```

Example 5.6: Creating the collection of TIT-family items placed in the specification that have the risk above the assumed acceptance level, for example > 35% (percentage measure was applied in considered design).

```
Threats
self.tITItem->select(riskValueAssess()>0.35)
□
```

The last example needs extra comment concerning the risk estimation. The *riskValueAssess()* operation always concerns a threat item having the *`DAgrGeneric`* assigned, which ought to have the asset value declared.

6. SECURITY OBJECTIVES ELABORATION

6.1. Security objectives section of the Security Target defined by the standard

The third main development stage is the establishment of the TOE security objectives. The standard says that the objectives provide a concise statement of the intended response to the security problem [38], indicating the extent to which the security needs, defined by the security environment, are addressed by the TOE and its environment. This is the central point of the whole ST or PP, a kind of bridge between security concerns and precise security specification based on CC components. The trade-off concerning responsibility for security between the TOE and its environment (using IT and non-IT aspects) is very important there because it implies the nature and cost of the developed product. The concept included in the standard can be expressed as the class diagram, shown in the Fig. 6.1. It will be the basis for further developed models presented in this monograph, concerning the security objectives elaboration methodology and the specification means.

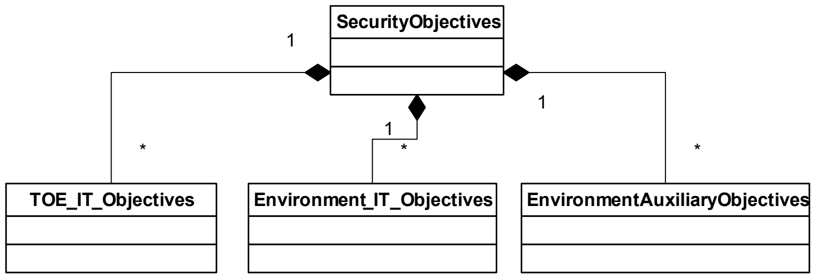


Fig. 6.1. TOE security objectives – the CC defined basic elements
Rys. 6.1. Cele zabezpieczeń przedmiotu oceny (TOE) – elementy zdefiniowane w standardzie Wspólne kryteria

6.2. Security objectives data model

For specification purposes different classes (including abstract ones) of security objectives should be distinguished (Fig. 6.2). The proposed objectives can be threats-, OSPs- or assumptions-driven. They concern the *Ogr4Tgr*, *Ogr4Pgr*, *Ogr4Agr* security association classes respectively (Fig. 3.18). The developers-defined auxiliary objectives can be driven in the same way.

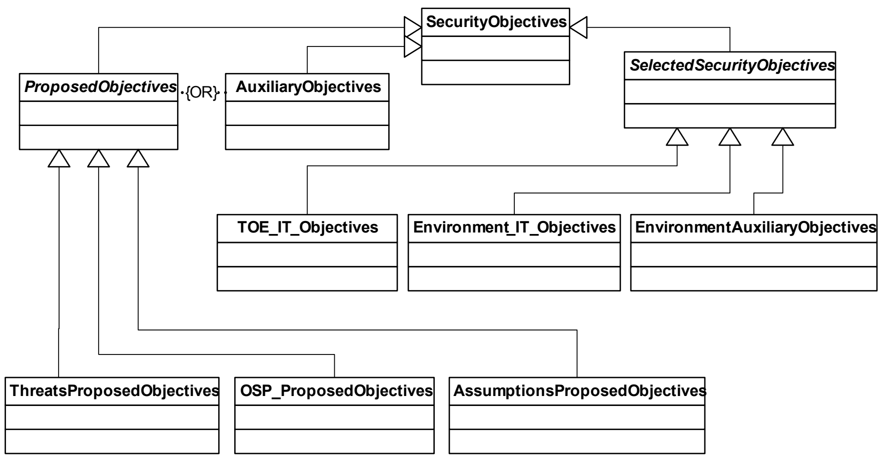

SecurityObjectives
ProposedObjectives
{OR}
AuxiliaryObjectives
SelectedSecurityObjectives
TOE_IT_Objectives
Environment_IT_Objectives
EnvironmentAuxiliaryObjectives
ThreatsProposedObjectives
OSP_ProposedObjectives
AssumptionsProposedObjectives

Fig. 6.2. Inherited classes expressing TOE security objectives
Rys. 6.2. Klasy dziedziczone, przedstawiające cele zabezpieczeń przedmiotu oceny (TOE)

The detailed class diagram was developed (Fig. 6.3) on the basis of the CC standard defined elements (Fig. 6.1) and the assumed types of security objectives (Fig. 6.2). The diagram presents more information concerning the modelling of any aspects of the TOE security objectives. As it was mentioned in the previous chapter, the `TOESecEnvElaboration` class is responsible for the security environment data structure workout. On the basis of this structure, the elements of the security objectives specification are elaborated. The `TOESecObjElaboration` class is responsible for that.

Different class attributes were indicated. Two Boolean-type flags: *dealingTOE, dealingEnviron*, assigned previously (i.e. during the security environment elaboration) to any identified threat or OSP, are now transferred to the assigned security objectives, although during subsequent security objectives trade-off the flags can be ultimately changed, allocating intentional response to the security concern. Each of the assigned security objectives should be justified – why it is needed, what is covered, what gaps still exist, what additional concerns are covered. The *coverage* flag should be false until the concern is not fully covered. All this information is sampled for a detailed analysis during the rationale process.

Three additional Boolean-type flags characterize the ways in which the security objectives affect a given threat or support a given OSP, i.e.: *detective*, *corrective* or *preventive*.

Any threat, OSP or assumption has its own list of proposed security objectives, covering them by default. This mechanism is an element of developers' decision support while safeguarding measures are selected. The developers can define their own security objectives (auxiliary) when the existing ones are inadequate.

There are other Boolean-type flags pointing to the security objective origin: i.e.: *threatDerived, ospDerived or assumptDerived*, used during the rationale process.

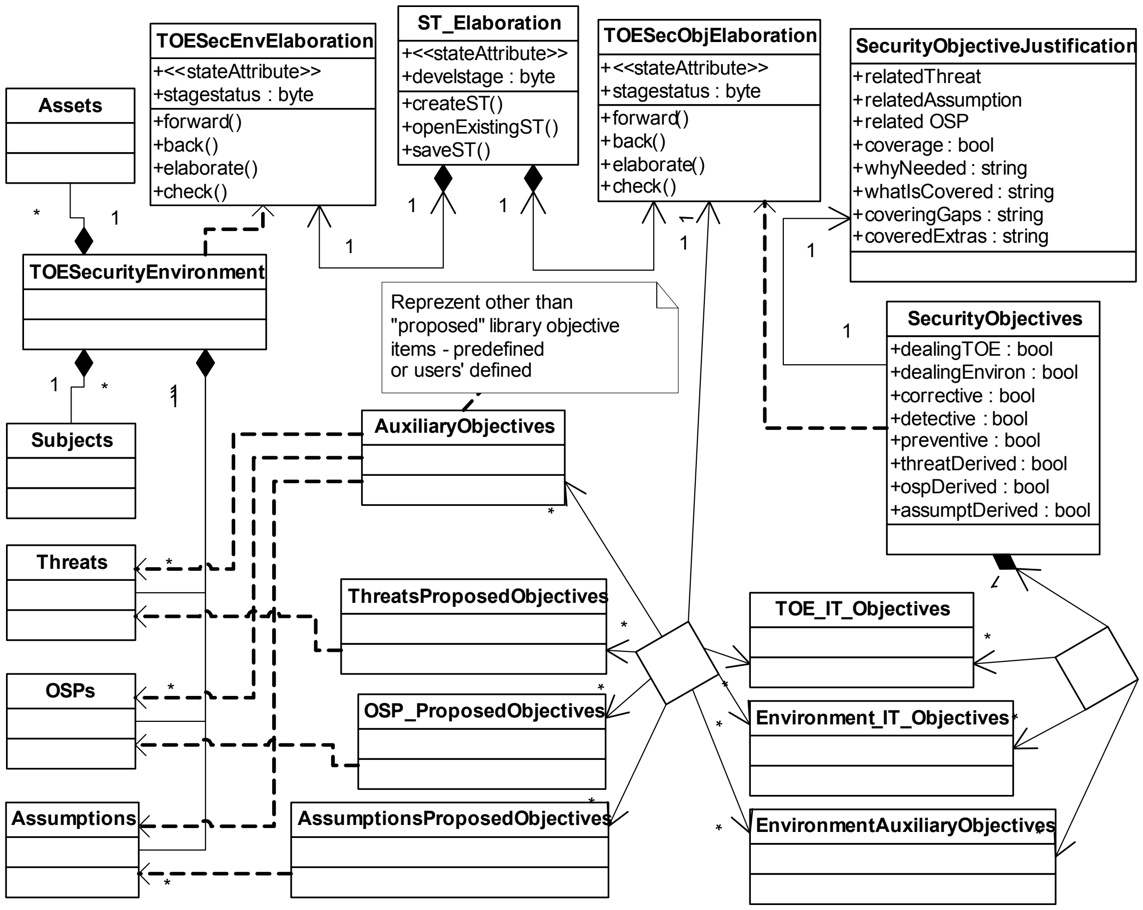


Fig. 6.3. *TOE security objectives elaboration (the right side) on the basis of the TOE security environment (the left side) – the assumed static structure diagram*
Rys. 6.3. *Wypracowanie celów zabezpieczeń TOE (strona prawa) na podstawie otoczenia zabezpieczeń (strona lewa) – założona struktura statyczna*

The key issue for the model presented there is using the risk assessment results and performing a coverage analysis with justification for the security concerns specified in the security environment. Every threat has additional data concerning the risk assigned, i.e. the asset value and the event likelihood, allowing to assess the risk value in a simple way and to order all scenarios according to these values, thus getting clear directions for security objectives selection.

The "proposed" objectives concern the `Ogr4Tgr`, `Ogr4Pgr`, `Ogr4Agr` security associations with *assocstat*=#PROPOSED (Fig. 3.17). These associations express the default assignments of security objectives to the threat-, OSP-, and assumption-families respectively. Due to the complicity of relationships between generic families of different types, all proposed relations will be presented on separate diagrams. Each of them can be considered as the refinement of the proper security association.

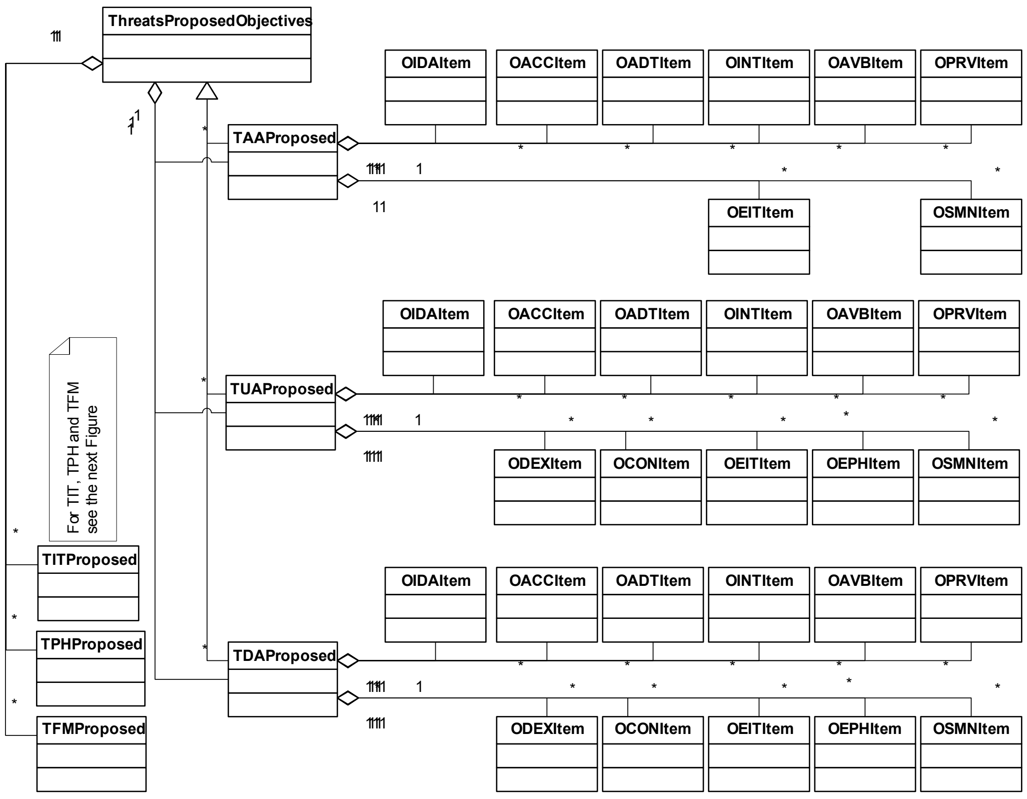


Fig. 6.4. Threats-proposed security objectives – families: TAA, TUA, TDA
Rys. 6.4. Cele zabezpieczeń sugerowane na podstawie rodzajów zagrożeń – rodziny typu: TAA, TUA, TDA

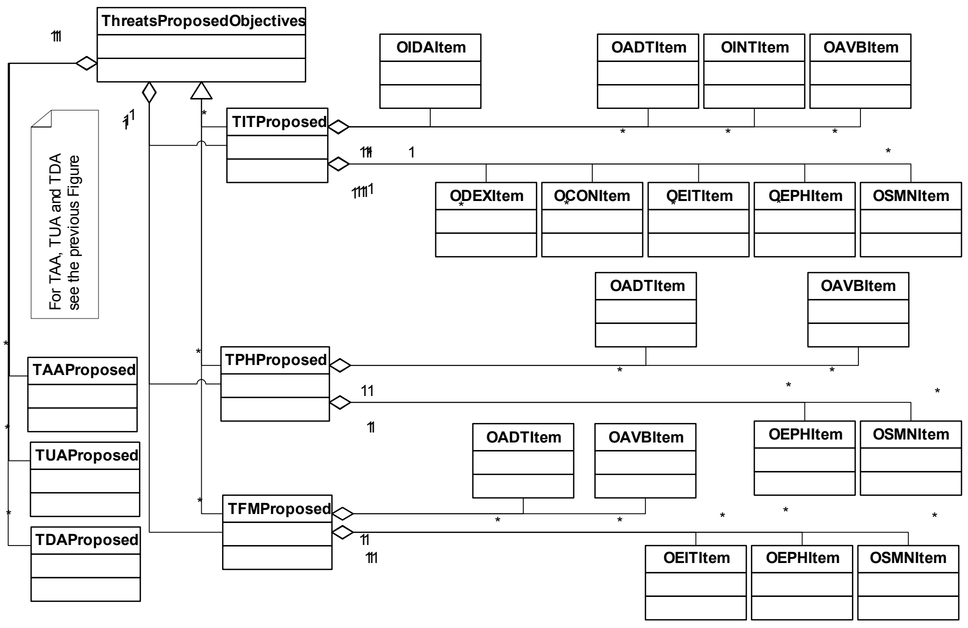


Fig. 6.5. Threats-proposed security objectives – families: TIT, TPH, TFM
Rys. 6.5. Cele zabezpieczeń sugerowane na podstawie rodzajów zagrożeń – rodziny typu: TIT, TPH, TFM

Every threat family, expressed by its generic, has a proposed security objectives subset defined (Fig. 6.4, Fig. 6.5), allowing developers' direct decision support. Please note the specific generic types allowed for the given threat family.

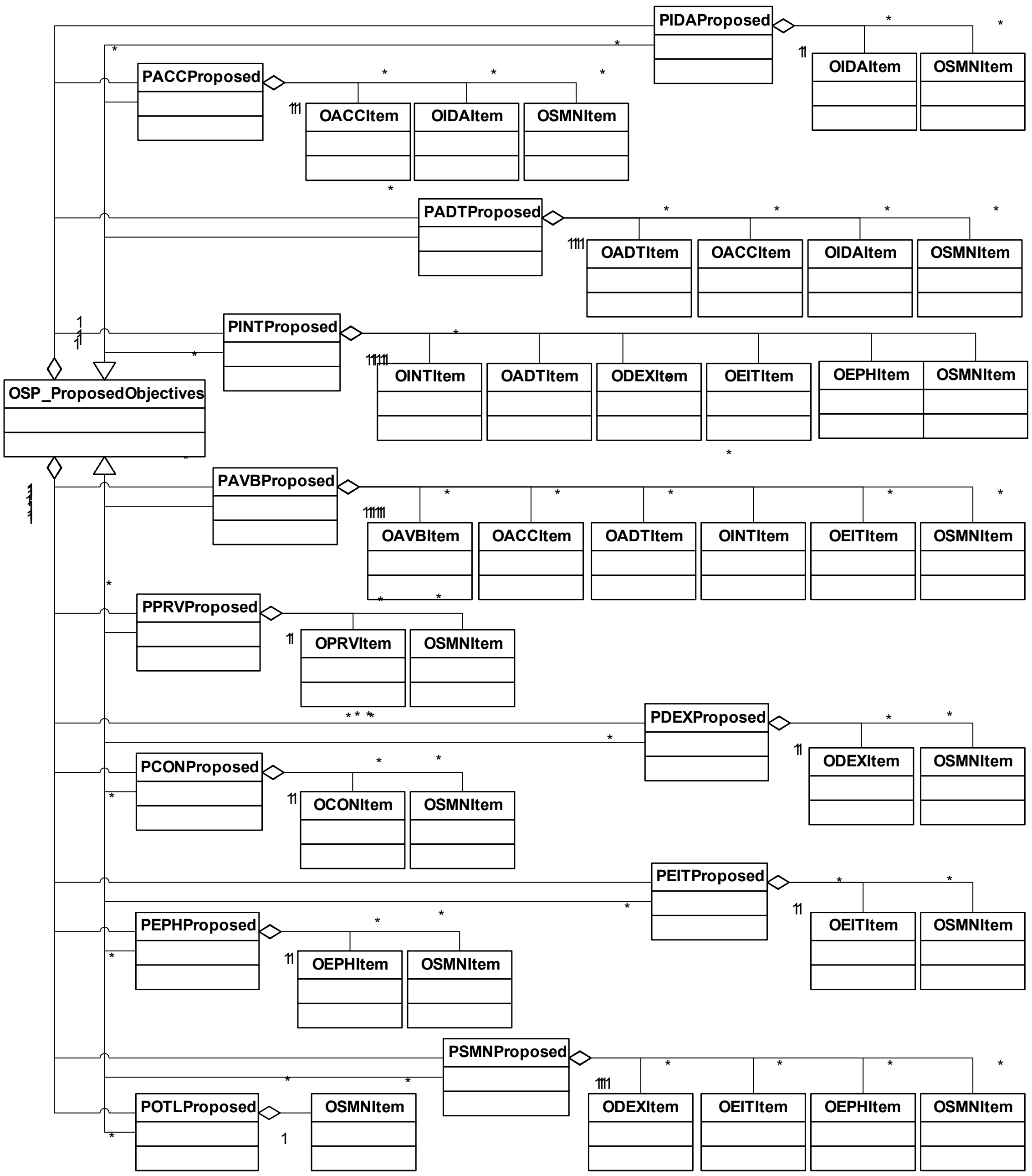


Fig. 6.6. The OSP-driven security objectives
Rys. 6.6. Cele zabezpieczeń sugerowane na podstawie reguł polityki bezpieczeństwa

The OSP-driven security objectives are presented in the Fig. 6.6. Every OSP family can be covered by the right security objectives items, selected after the security concern analysis. Please refer to the `Ogr4Pgr` security association which is expressed there in a more detailed way (on the generic family level).

The Fig. 6.7 presents security objectives influenced by the assumptions. Please note the right generic item family assigned to the given assumption generic family. This assignment is the refinement of the `Ogr4Agr` security association.

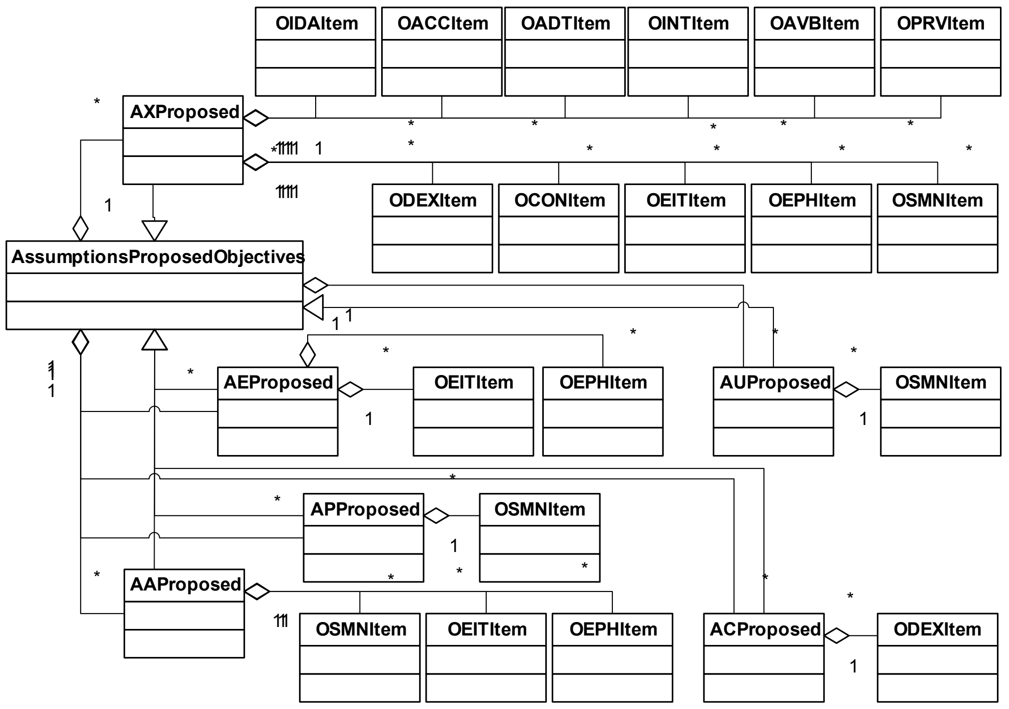


Fig. 6.7. The assumption-driven security objectives
Rys. 6.7. Cele zabezpieczeń sugerowane na podstawie założeń otoczenia zabepieczeń

The security objectives assignments, including those for threats and OSPs, refer to basic generics chains implemented in the generics library.

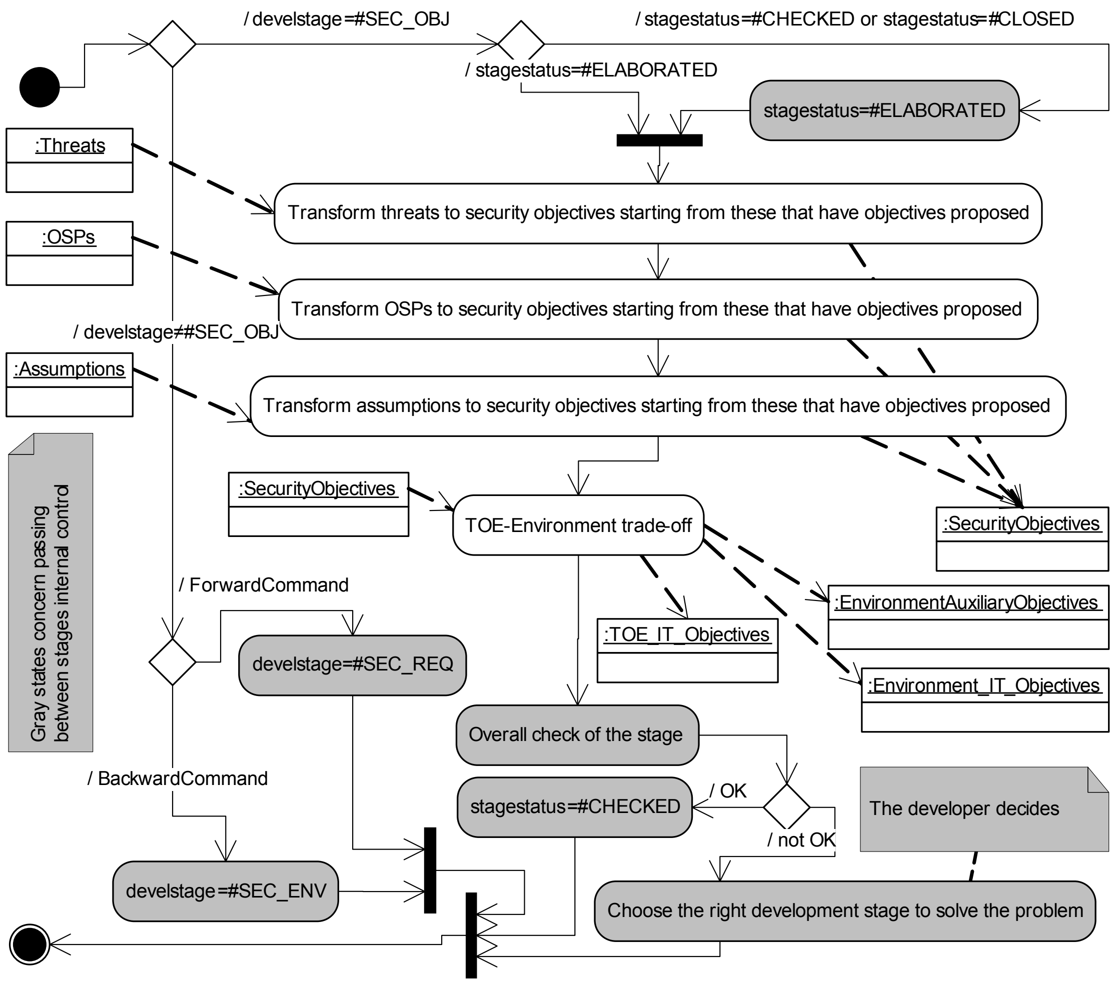
6.3. Security objectives specification workout

The general elaboration scheme, as a UML activity diagram, was shown in the Fig. 6.8 and represents the behaviour of the `TOESecObjElaboration` class. Four main steps are distinguished:

- working out security objectives specification implied by threat specification – starting from the threat is preferred by [60], generally allowing more optimal design,
- adding security objectives implied by OSPs (sometimes may be more convenient),
- finishing with adding some objectives influenced by assumptions,
- partitioning elaborated security objectives between the TOE and/or its environment.

Please note that threat specification has the highest priority, although when it does not exist all security objectives may be elaborated on the OSPs and/or assumptions basis. For the security objectives selection the following rules are assumed. First, the developer should merge or join the security objective proposed for the security item (i.e. the threat, the OSP, or the assumption), if existing, to the previously selected security objectives. Merging is possible when the proposed item overlaps the previously selected one, while joining is equal to adding separate proposed issues to the selected items.

This allows to get more compact specifications. Then the developer uses predefined generics (modifying them if needed) and joins them to the selected ones. If an adequate generic does not exist, a user-defined one is created and then added to the library resources and to the selected security objectives.



/ develstage=#SEC_OBJ
/ stagestatus=#CHECKED or stagestatus=#CLOSED
/ stagestatus=#ELABORATED
stagestatus=#ELABORATED
:Threats
Transform threats to security objectives starting from these that have objectives proposed
:OSPs
Transform OSPs to security objectives starting from these that have objectives proposed
/ develstage≠#SEC_OBJ
:Assumptions
Transform assumptions to security objectives starting from these that have objectives proposed
Gray states concern passing between stages internal control
:SecurityObjectives
TOE-Environment trade-off
:SecurityObjectives
:EnvironmentAuxiliaryObjectives
/ ForwardCommand
develstage=#SEC_REQ
:TOE_IT_Objectives
:Environment_IT_Objectives
Overall check of the stage
/ BackwardCommand
stagestatus=#CHECKED
/ OK
/ not OK
The developer decides
develstage=#SEC_ENV
Choose the right development stage to solve the problem

Fig. 6.8. TOE security objectives elaboration – the general activity diagram
Rys. 6.8. Wypracowanie celów zabezpieczeń TOE – nadrzędny diagram czynności

Each of the above mentioned steps will be shown on a separate activity diagram. The first subordinated diagram is focused on the threats basis selection (Fig. 6.9). This is the key part of the elaboration, encompassing almost all security concerns, because for any threat involved, supporting or redundant (should be removed) OSPs are analyzed with respect to risk scenarios.

These are the three main factors that should be considered while objectives are selected. This method allows to minimize the quantity of security objectives, no matter whether they are expressed by threats or/and by OSPs.

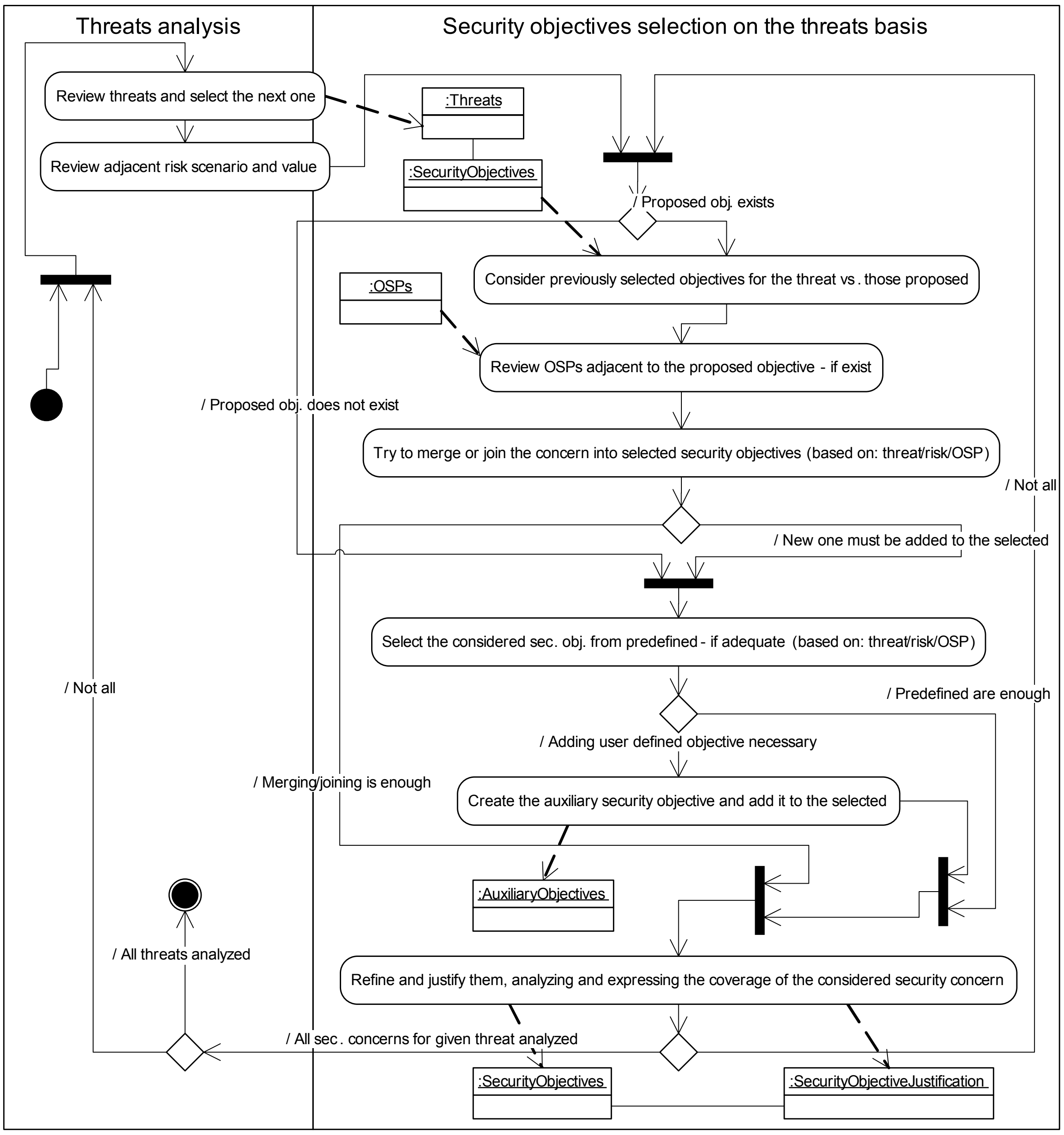

Threats analysis
Security objectives selection on the threats basis
Review threats and select the next one
Review adjacent risk scenario and value
:Threats
:SecurityObjectives
/ Proposed obj. exists
:OSPs
Consider previously selected objectives for the threat vs. those proposed
Review OSPs adjacent to the proposed objective - if exist
/ Proposed obj. does not exist
Try to merge or join the concern into selected security objectives (based on: threat/risk/OSP)
/ Not all
/ New one must be added to the selected
Select the considered sec. obj. from predefined - if adequate (based on: threat/risk/OSP)
/ Not all
/ Predefined are enough
/ Adding user defined objective necessary
/ Merging/joining is enough
Create the auxiliary security objective and add it to the selected
:AuxiliaryObjectives
/ All threats analyzed
Refine and justify them, analyzing and expressing the coverage of the considered security concern
/ All sec. concerns for given threat analyzed
:SecurityObjectives
:SecurityObjectiveJustification

Fig. 6.9. TOE security objectives elaboration – objectives selection on the threats basis
Rys. 6.9. Wypracowanie celów zabezpieczeń TOE – selekcja celów według zagrożeń

Finally, security objectives refinement and justification are performed, allowing not fully covered entities. All aspects of the identified security needs expressed by the TOE threat specification ought to be suitably addressed by the security objectives.

The second subordinated diagram presents the identified objectives using the OSPs specification (Fig. 6.10). At the beginning it is assumed that the threat specification was previously processed or it does not exist at all. In this case OSPs-expressed concerns are grouped too and if an adequate one does not exist, a new one can be defined – it is a general rule for all elaborations described there. In the end, refinement and justification are performed.

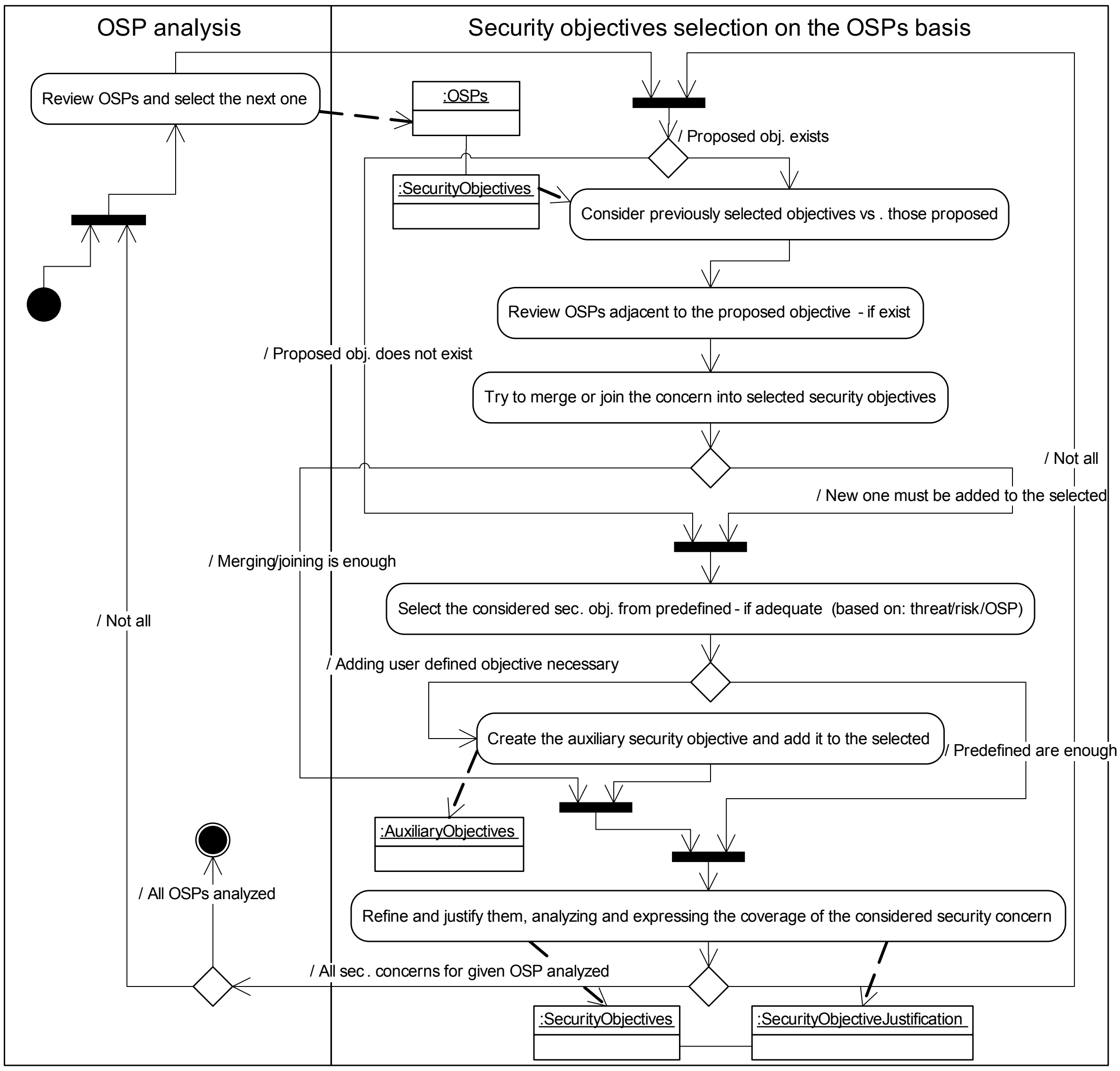

OSP analysis
Security objectives selection on the OSPs basis
Review OSPs and select the next one
:OSPs
/ Proposed obj. exists
:SecurityObjectives
Consider previously selected objectives vs . those proposed
Review OSPs adjacent to the proposed objective - if exist
/ Proposed obj. does not exist
Try to merge or join the concern into selected security objectives
/ Not all
/ New one must be added to the selected
/ Merging/joining is enough
/ Not all
Select the considered sec. obj. from predefined - if adequate (based on: threat/risk/OSP)
/ Adding user defined objective necessary
Create the auxiliary security objective and add it to the selected
/ Predefined are enough
:AuxiliaryObjectives
/ All OSPs analyzed
Refine and justify them, analyzing and expressing the coverage of the considered security concern
/ All sec . concerns for given OSP analyzed
:SecurityObjectives
:SecurityObjectiveJustification

Fig. 6.10. TOE security objectives elaboration – objectives selection on the OSPs basis
Rys. 6.10. Wypracowanie celów zabezpieczeń TOE – selekcja celów według reguł polityki bezpieczeństwa

All aspects of the identified security needs expressed by the OSP specification should be suitably addressed by the security objectives. The security objectives specification should be finally supplemented by the TOE environmental requirements, expressed by the assumptions (Fig. 6.11) – it is the whole specification refinement. All aspects of the identified security needs expressed by the TOE threat or OSP specification, suitably addressed by the security objectives, ought to be upheld by the assumptions.

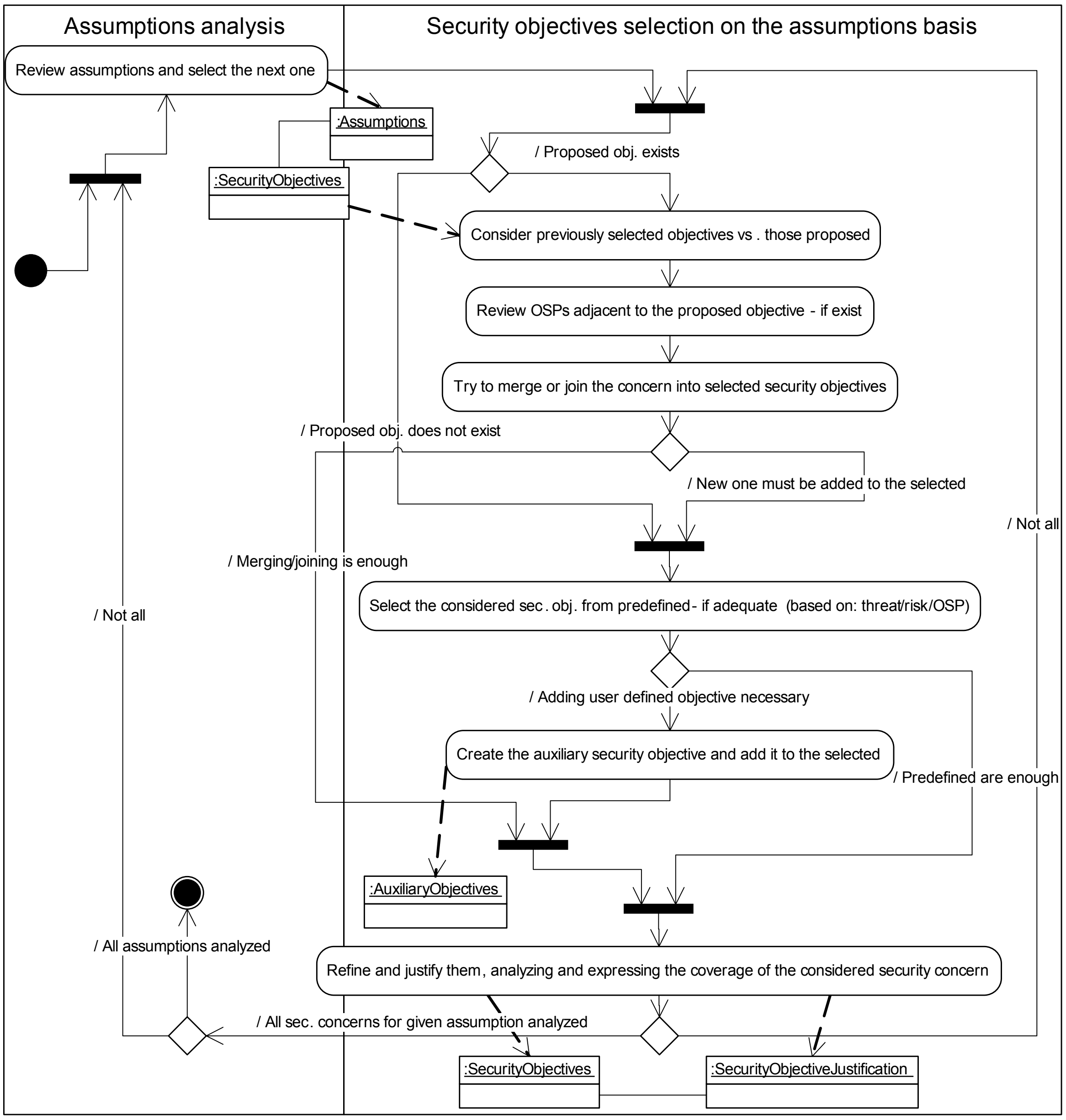

Assumptions analysis
Security objectives selection on the assumptions basis
Review assumptions and select the next one
:Assumptions
:SecurityObjectives
/ Proposed obj. exists
Consider previously selected objectives vs . those proposed
Review OSPs adjacent to the proposed objective - if exist
Try to merge or join the concern into selected security objectives
/ Proposed obj. does not exist
/ New one must be added to the selected
/ Not all
/ Merging/joining is enough
/ Not all
Select the considered sec. obj. from predefined - if adequate (based on: threat/risk/OSP)
/ Adding user defined objective necessary
Create the auxiliary security objective and add it to the selected
/ Predefined are enough
:AuxiliaryObjectives
/ All assumptions analyzed
Refine and justify them, analyzing and expressing the coverage of the considered security concern
/ All sec. concerns for given assumption analyzed
:SecurityObjectives
:SecurityObjectiveJustification

Fig. 6.11. TOE security objectives elaboration – objectives selection on the assumptions basis
Rys. 6.11. Wypracowanie celów zabezpieczeń TOE – selekcja celów według założeń otoczenia

The elaborated security objectives specification includes items assigned by default to the TOE and/or its environment (Fig. 6.12) according to the mentioned Boolean flags. The developer is able to optimize the specification assigning the responsibility for the intended security problems response between the TOE and/or its environment, and deciding about the functionality and cost of the designed TOE.

Please note that TOE security objectives will be implemented and evaluated, but those dealing with the environment will be satisfied by the environment (usually in the organizational way) and will be considered as dogmas during evaluations. In the first case the developer can achieve the TOE with rich functionality, working automatically, probably more reliable but more costly. In the second case he/she can obtain products that need permanent personal assistance, have inconvenient maintenance, but are considerably cheaper. This trade-off depends indirectly on the declared EAL. In any case, two kinds of the environmental objectives should support the TOE objectives.

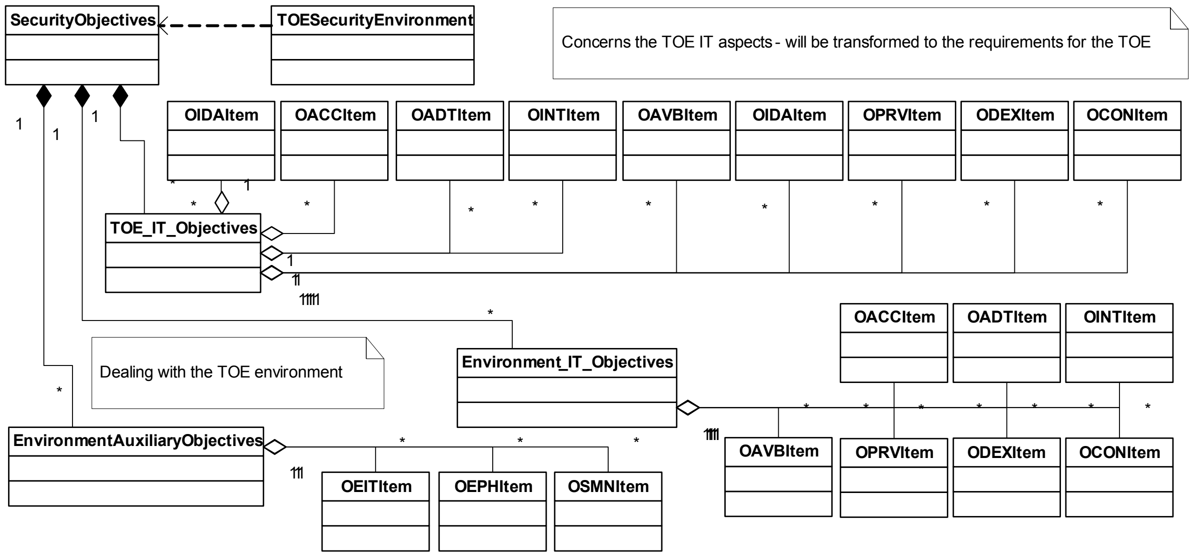


Fig. 6.12. TOE security objectives elaboration – objectives partitioning between the TOE and/or its environment
Rys. 6.12. Wypracowanie celów zabezpieczeń TOE – rozdział celów pomiędzy TOE lub jego otoczenie

The elaborated security objectives specification is the basis for the security requirements elaboration. Every security objective has its proposed security requirements assigned, supporting developers' decisions.

6.4. Formal approach to the security objectives specification

In the same way as for the security environment, some formalisms for the security objectives elaboration will be proposed. The security objectives specification can be expressed as the triplet of sets.

Definition 6.1: TOE security objectives.

The TOE security objectives is the triplet of sets:

$S_{SecObj} = <S_{SecObjItems}, S_{SecObjPars}, S_{SecObjAssocs}>$.

1. The first element of the triplet, $S_{SecObjItems}$ is the sum of three disjointed[15] sets of generics:

$S_{SecObjItems} = S_{TOE\_ITObjectives} \cup S_{EnvirITObjectives} \cup S_{EnvirAuxObjectives}$ and

$S_{TOE\_ITObjectives} \cap S_{EnvirITObjectives} \cap S_{EnvirAuxObjectives} = \varnothing$, where:

[15] This condition ought to be satisfied at the end of work on security objectives specification. At the beginning, IT security objectives for the TOE and environment usually have common items. It is a post-condition, not precondition, with respect to `TOESecObjElaboration.elaborate()` (Fig. 6.3).

$S_{TOE\_ITObjectives} = \{\texttt{OgrGeneric} \mid \texttt{OgrGeneric}.\mathit{assignstat}{=}\#\mathrm{ASSIGNED}\}$,
$S_{EnvirITObjectives} = \{\texttt{OACCGen} \mid \texttt{OACCGen}.\mathit{assignstat}{=}\#\mathrm{ASSIGNED}\} \cup$
$\{\texttt{OADTGen} \mid \texttt{OADTGen}.\mathit{assignstat}{=}\#\mathrm{ASSIGNED}\} \cup$
$\{\texttt{OINTGen} \mid \texttt{OINTGen}.\mathit{assignstat}{=}\#\mathrm{ASSIGNED}\} \cup$
$\{\texttt{OAVBGen} \mid \texttt{OAVBGen}.\mathit{assignstat}{=}\#\mathrm{ASSIGNED}\} \cup$
$\{\texttt{OPRVGen} \mid \texttt{OPRVGen}.\mathit{assignstat}{=}\#\mathrm{ASSIGNED}\} \cup$
$\{\texttt{ODEXGen} \mid \texttt{ODEXGen}.\mathit{assignstat}{=}\#\mathrm{ASSIGNED}\} \cup$
$\{\texttt{OCONGen} \mid \texttt{OCONGen}.\mathit{assignstat}{=}\#\mathrm{ASSIGNED}\}$.
$S_{EnvirAuxObjectives} = \{\texttt{OEITGen} \mid \texttt{OEITGen}.\mathit{assignstat}{=}\#\mathrm{ASSIGNED}\} \cup$
$\{\texttt{OEPHGen} \mid \texttt{OEPHGen}.\mathit{assignstat}{=}\#\mathrm{ASSIGNED}\} \cup$
$\{\texttt{OSMNGen} \mid \texttt{OSMNGen}.\mathit{assignstat}{=}\#\mathrm{ASSIGNED}\}$.

2. The second element of the triplet, $S_{SecObjPars}$ is the sum of two sets of parameterization associations:

$S_{SecObjPars} \subseteq \mathsf{GenParAssoc}$ and $S_{SecObjPars} = \{\texttt{ParamDA4O}, \texttt{ParamS4O}\}$.

3. The final element of the triplet, $S_{SecObjAssocs}$ is the sum of three sets of associations dealing with the mapping of the security objectives to the threats, OSPs and assumptions items:

$S_{SecObjAssocs} \subseteq \mathsf{SecAssoc}$ and $S_{SecObjAssocs} = \{\texttt{Ogr4Tgr}, \texttt{Ogr4Pgr}, \texttt{Ogr4Agr}\}$.

□

Please refer to the Fig. 6.3, 6.12, 3.15 and 3.18 which exemplify this definition. The parameters can be left empty, i.e. uncompleted or should be properly assigned. Assuming that the types of parameter attributes are: *paramDAgr:* `DAgrGeneric` and *paramSgr:* `SgrGeneric`, this condition can be easily expressed as the OCL invariants:

```
ParamDA4O
OgrGeneric.paramDAgr=DAgrGeneric or DAgrGeneric->isEmpty
ParamS4O
OgrGeneric.paramSgr=SgrGeneric or SgrGeneric->isEmpty
```

□

Let us consider a very general definition of the security objectives elaboration process that summarizes all activities expressed semiformally in this chapter:

Definition 6.2: TOE security objectives elaboration.

The TOE security objectives elaboration is the transformation of $S_{TOESecEnv}$ to S_{SecObj}, i.e. transformation of $S_{TOESecEnvItems}$ with their parameters $S_{TOESecEnvPars}$, to $S_{SecObjItems}$ with their parameters $S_{SecObjPars}$, according to the mapping rules represented by the $S_{SecObjAssocs}$ associations.

□

Discussing the security environment specification (section 5.6) some examples of the OCL expressions, mainly operations on collections, were presented. They can be used for other IT security development stages too. For the security objectives elaboration stage the new element has appeared, i.e. the set of associations concerning mapping the security issues.

Example 6.1: Searching for the similar items preliminarily assigned to the TOE and the environment – the general checking before the trade-off activity (see Fig. 6.8).

At the beginning of the security objectives stage many security objectives items have both flags *dealingTOE*, *dealingEnviron* set, i.e. inherited from threats, OSPs or assumptions items.

This inconsistency of the model ought to be corrected by the rearrangement of the security objectives items, but firstly all these items should be identified.
The searching is restricted to the selected family of generics and the operation belongs to the `SecurityObjectives` class.

```
SecurityObjectives::search4Unresolved(p:GenFamily)
pre:
post: result=self.TOE_IT_Objectives.p->
    select(dealingTOE=true and dealingEnviron=true)->union
    (self.Environment_IT_Objectives.p->
    select(dealingTOE=true and dealingEnviron=true)->union
            (self.EnvironmentAuxiliaryObjectives.p->
            select(dealingTOE=true and dealingEnviron=true)
            )
    )
```

□

Example 6.2: Finding the repeating security issues in two sets of items – the final checking after the trade-off activity.

These operations search for identical items belonging to different sets of the security objectives and can be applied to the final specification checking during or after the trade-off activity.
As it was mentioned earlier, at the beginning of the security objectives specification workout some items, usually derived from the "proposed", are placed into both the TOE IT objectives and Environment IT objectives sets, or into both the Environment IT objectives and Environment Auxiliary objectives sets.
It was assumed that the discussed two operations belong to the `SecurityObjectives` class.

```
SecurityObjectives::commonITObjectives(p:GenFamily)
pre:
post: result=self.TOE_IT_Objectives.p->intersection
    (self.Environment_IT_Objectives.p)

SecurityObjectives::commonEnvironObjectives(p:GenFamily)
pre:
post: result=self.Environment_IT_Objectives.p->intersection
    (self.EnvironmentAuxiliaryObjectives.p)
```

□

7. PREPARING SECURITY REQUIREMENTS

7.1. Security requirements specification according to the standard

The standard says that the following three main types of IT security requirements should be specified (Fig. 7.1):

- TOE security requirements – expressed with the use of CC components, evaluated and consisting of:
  - Security Functional Requirements (SFRs) for the TOE – expressed by functional components [39],
  - Security Assurance Requirements (SARs) for the TOE – expressed by assurance components [40];
- IT environment security requirements, non evaluated and considered as dogmas during the evaluation – expressed in two ways:
  - using functional and/or assurance CC components (for a well defined TOE IT environment);
  - using generics, when more general statements are enough (REIT-family or REPH-family generics);
- non-IT environment security requirements – supportive to the above mentioned, non evaluated and considered as dogmas, generally optional and expressed by the RENIT-family generics.

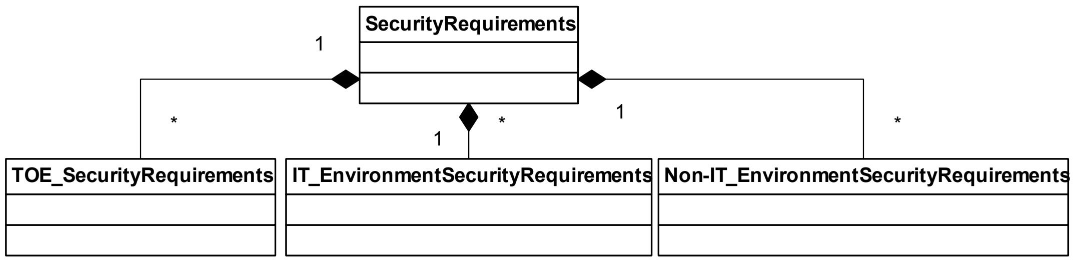

SecurityRequirements
1
*
1
*
1
*
TOE_SecurityRequirements
IT_EnvironmentSecurityRequirements
Non-IT_EnvironmentSecurityRequirements

Fig. 7.1. TOE security requirements – the CC defined basic elements
Rys. 7.1. Wymagania bezpieczeństwa dla przedmiotu oceny (TOE) – elementy określone w standardzie Wspólne kryteria

Besides, there are two main types of SFRs [60] distinguished:

- principal SFRs, which directly satisfy given TOE security objectives;

- supporting SFRs, which provide support to the principal SFRs, and hence indirectly help satisfy the relevant security objectives for the TOE.

Any SFR (principal or supportive) may have [39] a dependent SFR and/or SAR assigned. The security requirement elaboration process is rather well, but informally, described in the standard, so the monograph is focused on its semiformal modelling and implementation, assuring the consistency with other development stages encompassed by the IT Security Development Framework. The specification means for this stage are semiformally defined as the functional [39] and assurance [40] components.

7.2. Security requirements data model

The concept included in the standard (Fig. 7.1) will be the basis for further developed models presented in this monograph. Among the above mentioned types of requirements, implied directly or indirectly by the standard, additional groups of requirements are introduced (Fig. 7.2).

The requirements represent different data structures participating in the development process. Please note that some classes are abstract. The requirements grouping is specified in the Fig. 7.2. Generally, any component can be predefined and placed in CC catalogues, though some of them can be defined by developers (`UserDefinedReqs` class) to meet their specific needs. By analogy to the SFR, the Security-objectives-derived SARs and Supporting SARs are distinguished. To support developers' actions some of the requirements (CC components or RE-family generics) are proposed to solve a given security problem. Additionally, CC defined components dependencies are considered.

The CC components (proposed or not), user's defined components and the RE-group of generics represent the specifications means, though "selected" express the requirements for the given TOE.

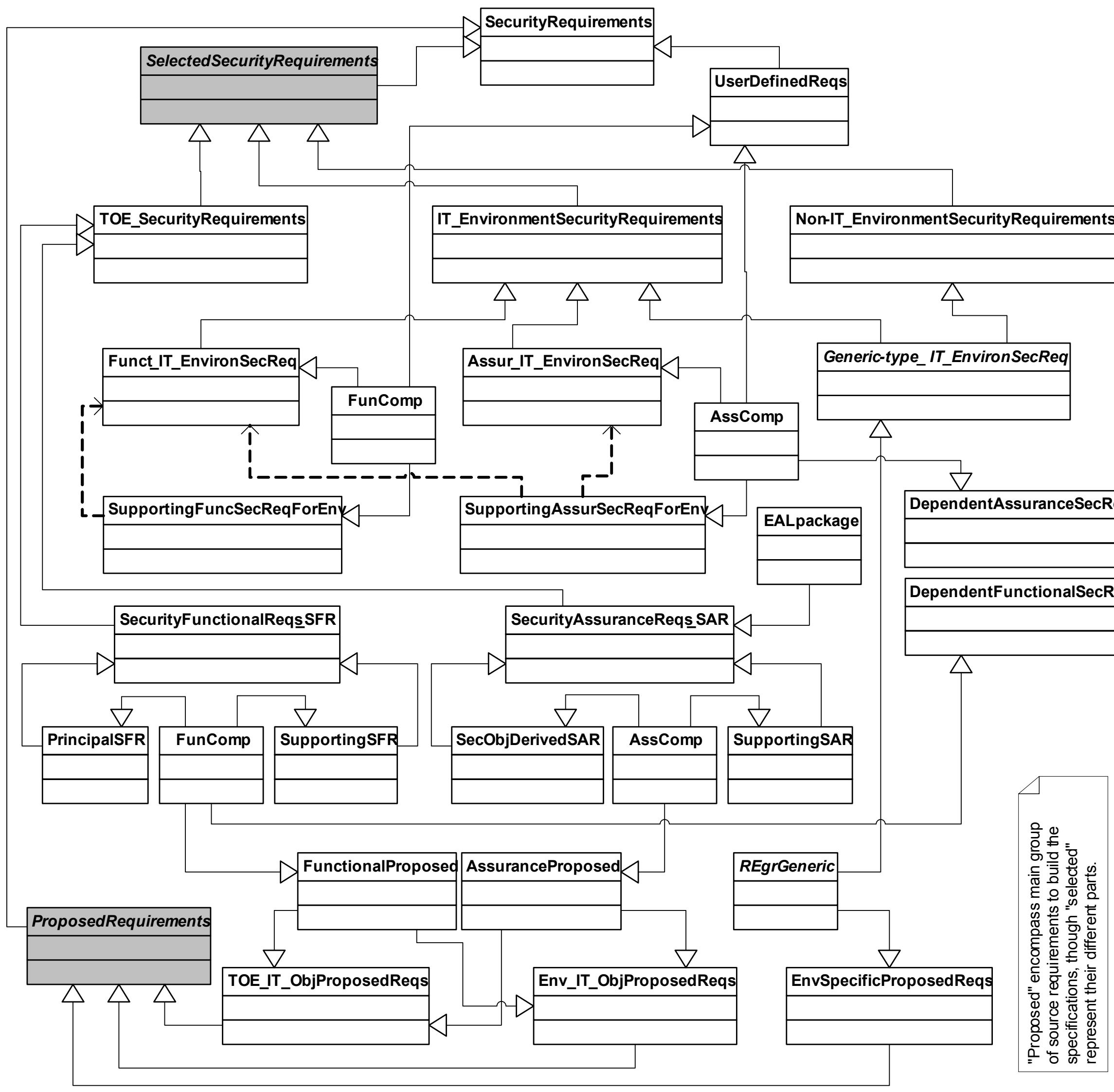

SecurityRequirements
SelectedSecurityRequirements
UserDefinedReqs
TOE_SecurityRequirements
IT_EnvironmentSecurityRequirements
Non-IT_EnvironmentSecurityRequirements
Funct_IT_EnvironSecReq
Assur_IT_EnvironSecReq
Generic-type_ IT_EnvironSecReq
FunComp
AssComp
SupportingFuncSecReqForEnv
SupportingAssurSecReqForEnv
EALpackage
DependentAssuranceSecReq
DependentFunctionalSecReq
SecurityFunctionalReqs_SFR
SecurityAssuranceReqs_SAR
PrincipalSFR
FunComp
SupportingSFR
SecObjDerivedSAR
AssComp
SupportingSAR
FunctionalProposed
AssuranceProposed
REgrGeneric
ProposedRequirements
TOE_IT_ObjProposedReqs
Env_IT_ObjProposedReqs
EnvSpecificProposedReqs
"Proposed" encompass main group of source requirements to build the specifications, though "selected" represent their different parts.

Fig. 7.2. Inherited classes expressing TOE security requirements
Rys. 7.2. Klasy dziedziczone, przedstawiające różne formy wymagań bezpieczeństwa dla przedmiotu oceny (TOE)

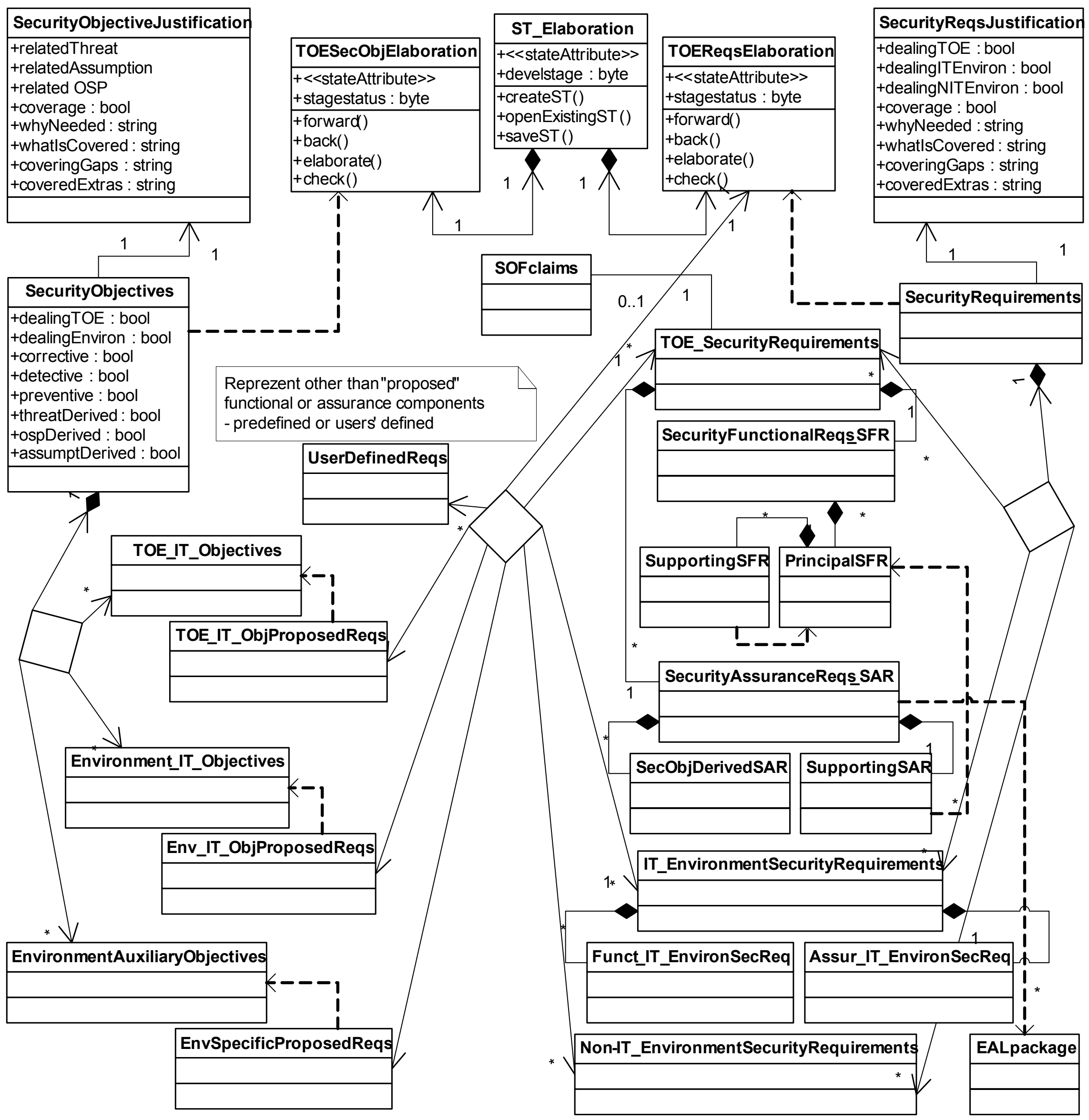


Fig. 7.3. The security requirements elaboration (the right side) on the basis of the TOE security objectives (the left side) – the assumed static structure diagram

Rys. 7.3. Wypracowanie wymagań bezpieczeństwa TOE (strona prawa) na podstawie celów zabezpieczeń (strona lewa) – założona struktura statyczna

The Fig. 7.3 presents more details on the main classes participating in the elaboration of the TOE and its environment security requirements.

In the previous chapter the behaviour of the `TOESecObjElaboration` class responsible for the security objectives data structure workout was discussed. On this basis, the elements of the security requirements specification are elaborated. The `TOESecReqsElaboration` class is responsible for that. The proposed, objectives-driven requirements (functional, assurance, environmental) concern the *`FunSec4Ogr`*, *`AssSec4Ogr`*, *`REgr4Ogr`* security association classes respectively (Fig. 3.19) with *assocstat* = #PROPOSED (Fig. 3.17).

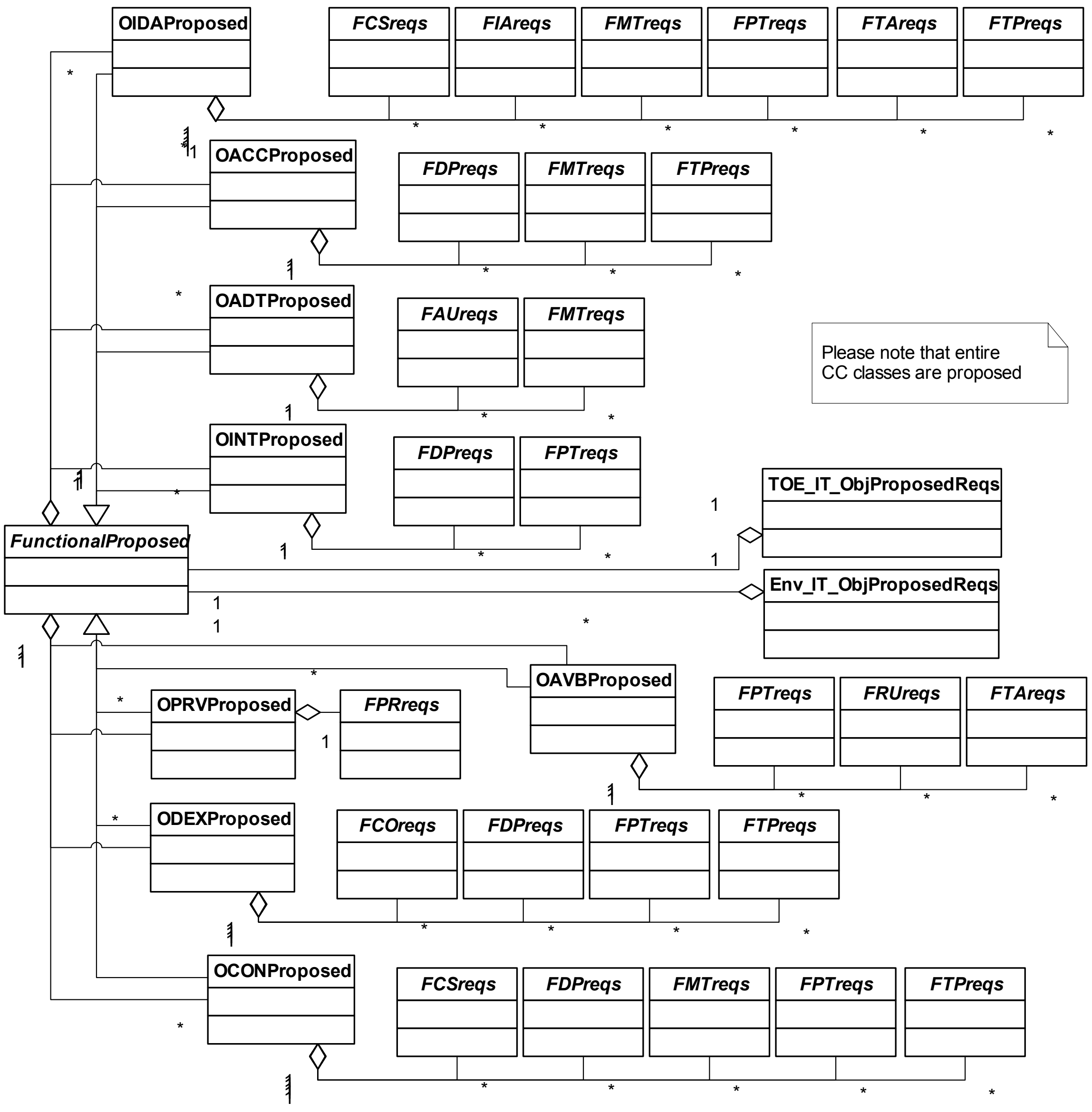


Fig. 7.4. Functional security requirements proposed by "well defined" security objectives (IT-type)
Rys. 7.4. Funkcjonalne wymagania bezpieczeństwa sugerowane na podstawie w pełni określonych celów zabezpieczeń (o charakterze informatycznym)

Attention should also be paid to security requirements justification structure, SOF claims and the EAL package, which is the set of the well balanced assurance components. The key issue of the computer-aided security development are requirements proposed to cover any security problem specified by the security objectives, grouped by their families. Any generic of a given family is transformed to a given subset of requirements represented by CC or user-defined components/generics. The Fig. 7.4 presents proposed functional requirements by "well defined" security objectives. They can be used for the TOE security requirements specification (SFR) and for its IT environment, when possible, i.e. the IT environment can be specified on a sufficiently detailed level (IT-type and "well defined").

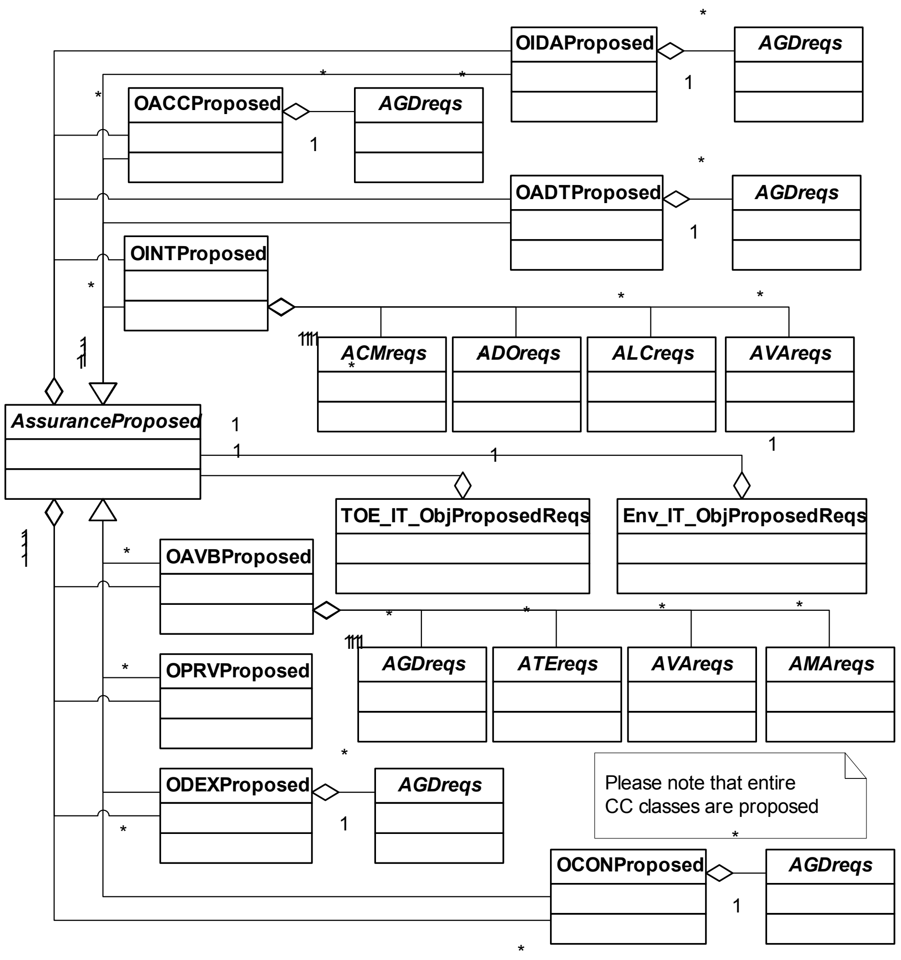


Fig. 7.5. Assurance security requirements proposed by "well defined" security objectives (IT-type)
Rys. 7.5. Wymagania bezpieczeństwa na uzasadnienie zaufania, sugerowane na podstawie w pełni określonych celów zabezpieczeń (o charakterze informatycznym)

The assurance security requirements, usually implied by the declared EAL level, can also be derived directly from the security objectives (Fig. 7.5). They assurance security requirements can be used for both the TOE and its environment, although for the TOE EALs are usually enough. For the TOE IT environment, however, requirements are often specified by the RE-family generics.

For the TOE environment requirements specification on a more general level, environment specific objectives families are defined (Fig. 7.6).

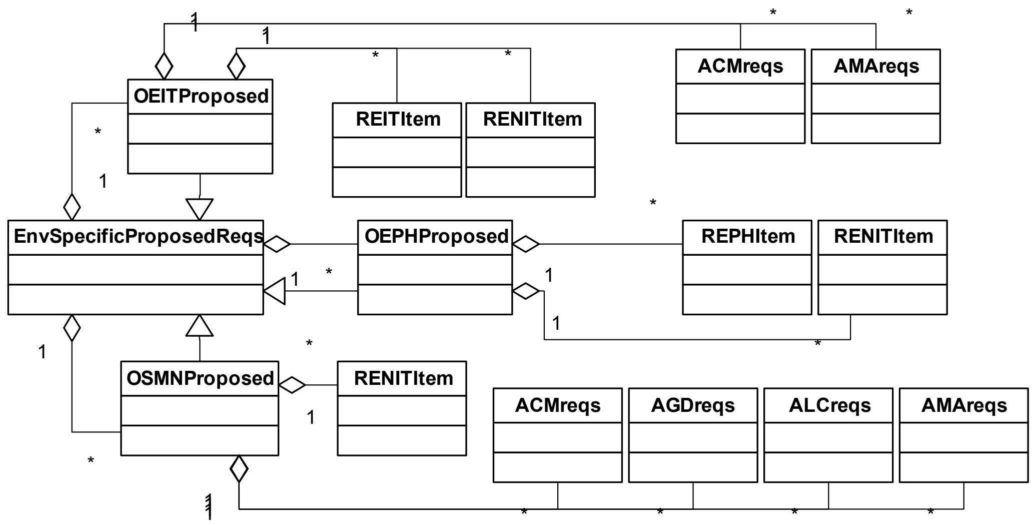


Fig. 7.6. Environment specific security requirements proposed by general purpose security objectives

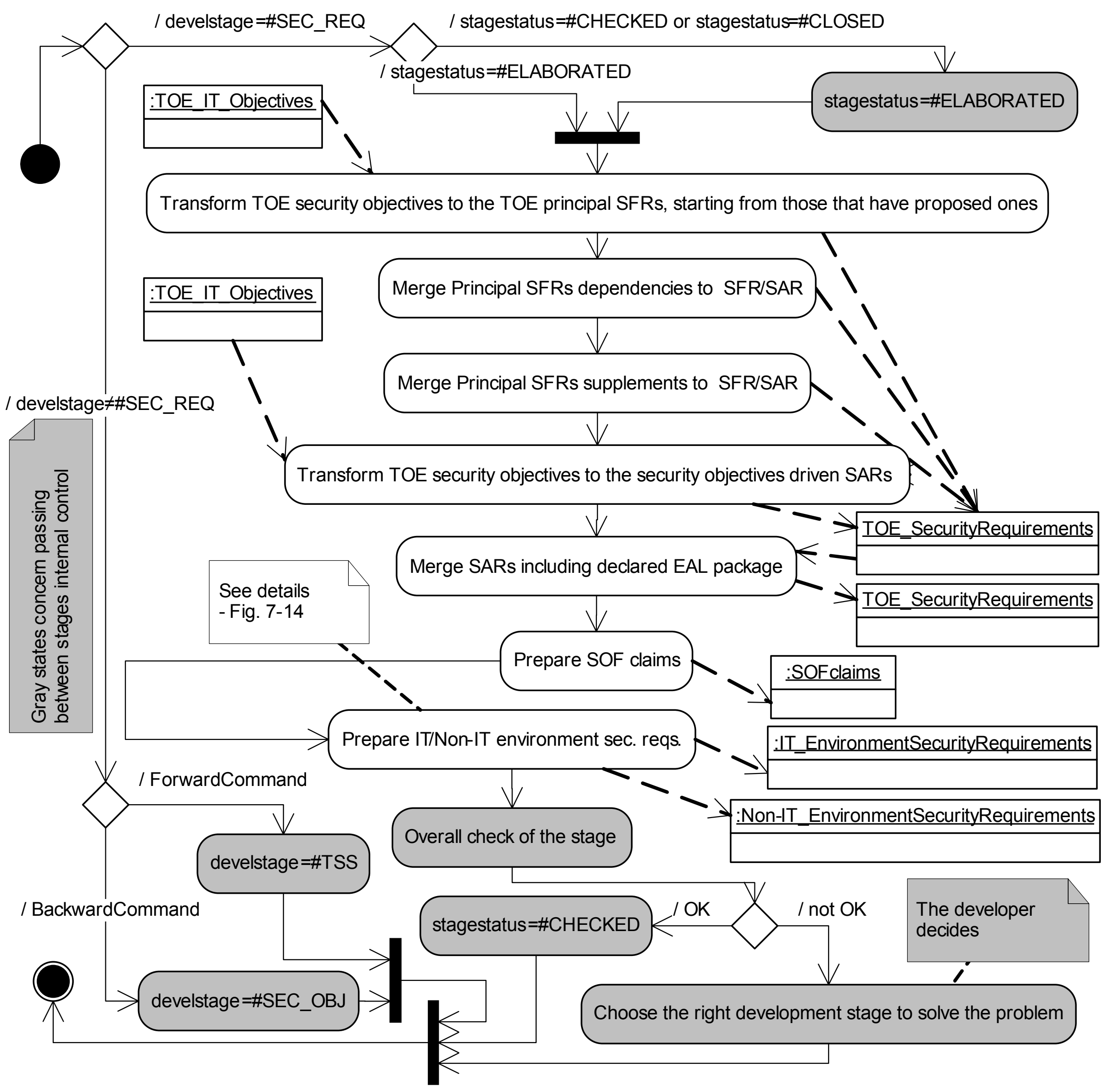
Rys. 7.6. Środowiskowe wymagania bezpieczeństwa sugerowane na podstawie celów zabezpieczeń ogólnego przeznaczenia

OEIT deals with IT aspects while OEPH with physical/technical aspects. They are associated with OSMN, representing all human concerns. They are transformed to IT or Non-IT environment security requirements respectively. Please note that some assurance CC components can also be derived from the general-purpose objectives.

7.3. Security requirements specification workout

This section presents the behaviour of the `TOEReqsElaboration` class. The general elaboration scheme of the security requirements, as a UML activity diagram, was shown in the Fig. 7.7 (seven steps – marked as “white” states).

The first six steps concerning the TOE security requirements are discussed in the section 7.3.1. The seventh step, dealing with the environment, is refined in the Fig. 7.14 and will be presented in the section 7.3.2.



/ develstage=#SEC_REQ
/ stagestatus=#CHECKED or stagestatus=#CLOSED
/ stagestatus=#ELABORATED
stagestatus=#ELABORATED
:TOE_IT_Objectives
Transform TOE security objectives to the TOE principal SFRs, starting from those that have proposed ones
:TOE_IT_Objectives
Merge Principal SFRs dependencies to SFR/SAR
Merge Principal SFRs supplements to SFR/SAR
/ develstage≠#SEC_REQ
Gray states concern passing between stages internal control
Transform TOE security objectives to the security objectives driven SARs
TOE_SecurityRequirements
Merge SARs including declared EAL package
TOE_SecurityRequirements
See details - Fig. 7-14
Prepare SOF claims
:SOFclaims
Prepare IT/Non-IT environment sec. reqs.
:IT_EnvironmentSecurityRequirements
/ ForwardCommand
:Non-IT_EnvironmentSecurityRequirements
develstage=#TSS
Overall check of the stage
/ BackwardCommand
stagestatus=#CHECKED
/ OK
/ not OK
The developer decides
develstage=#SEC_OBJ
Choose the right development stage to solve the problem

Fig. 7.7. TOE security requirements elaboration – the main activity diagram
Rys. 7.7. Wypracowanie wymagań bezpieczeństwa dla przedmiotu oceny (TOE) – nadrzędny diagram czynności

7.3.1. TOE security requirements

The six major activities of the TOE security requirements elaboration are the following:

- working out the functional principal security requirements derived directly from the security objectives – starting from those that have the proposed requirements to speed up the selection process,
- merging dependencies – due to CC components dependencies, all of them should be analyzed and accepted as supportive requirements or deliberately excluded,
- merging supplementary requirements as supporting SFRs and their dependencies to reinforce the principal SFRs,

- identifying assurance requirements derived directly from the security objectives,
- merging them with those which are implied by the declared EAL level,
- preparing SOF (Strength of Function) claims when permutation or probabilistic mechanisms and EAL>1 are assumed.

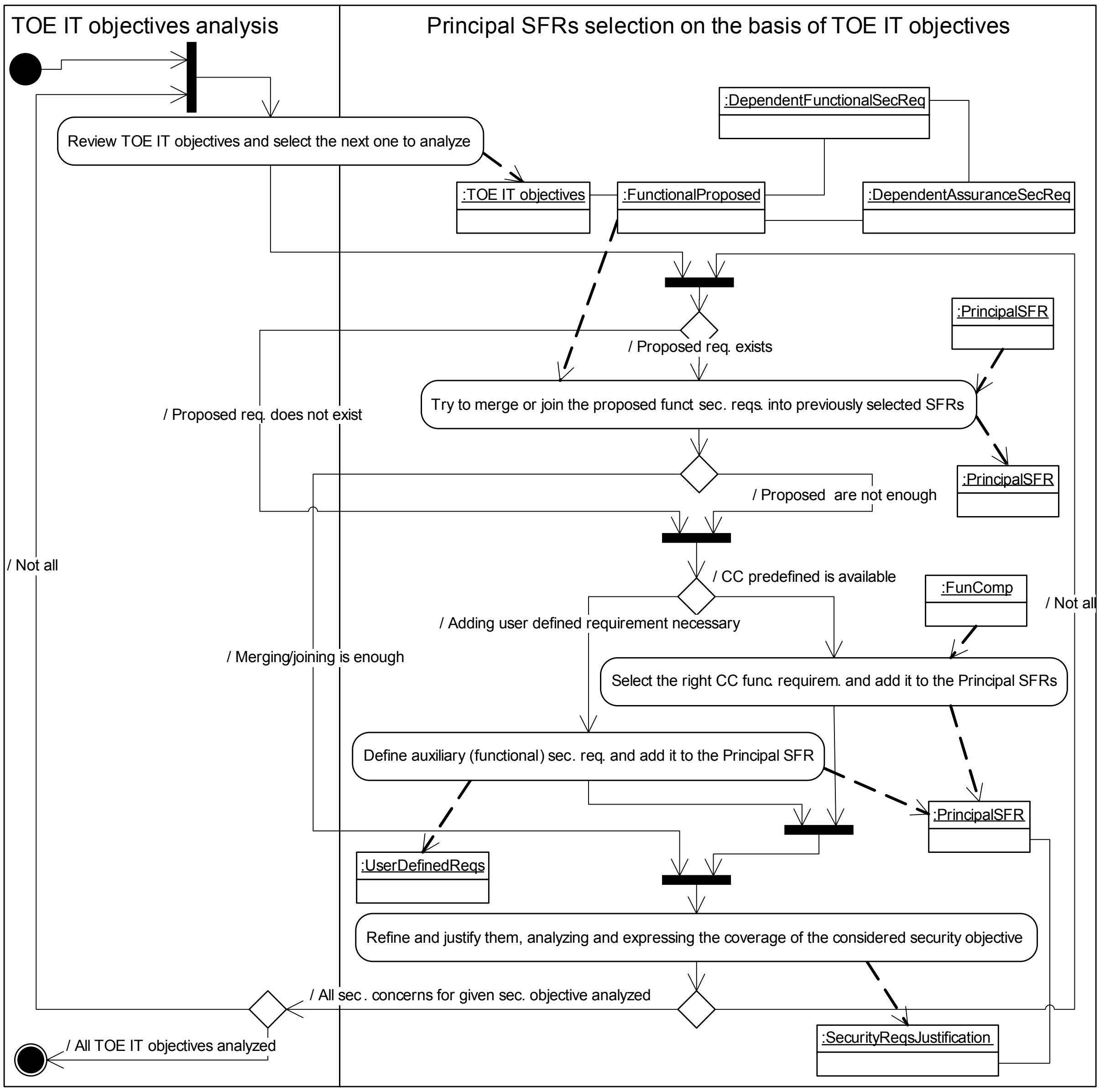

TOE IT objectives analysis
Principal SFRs selection on the basis of TOE IT objectives
Review TOE IT objectives and select the next one to analyze
:TOE IT objectives
:FunctionalProposed
:DependentFunctionalSecReq
:DependentAssuranceSecReq
:PrincipalSFR
/ Proposed req. exists
/ Proposed req. does not exist
Try to merge or join the proposed funct sec. reqs. into previously selected SFRs
:PrincipalSFR
/ Proposed are not enough
/ Not all
/ CC predefined is available
:FunComp
/ Not all
/ Adding user defined requirement necessary
/ Merging/joining is enough
Select the right CC func. requirem. and add it to the Principal SFRs
Define auxiliary (functional) sec. req. and add it to the Principal SFR
:PrincipalSFR
:UserDefinedReqs
Refine and justify them, analyzing and expressing the coverage of the considered security objective
/ All sec. concerns for given sec. objective analyzed
/ All TOE IT objectives analyzed
:SecurityReqsJustification

Fig. 7.8. TOE functional principal security requirements elaboration
Rys. 7.8. Wypracowanie podstawowych, funkcjonalnych wymagań bezpieczeństwa dla przedmiotu oceny (TOE)

Each activity will be shown on a separate diagram. For the later, and successful, rationale process the following goals should be achieved during this elaboration:

- the security objectives for the TOE should be suitably met by the identified IT security requirements (functional and assurance),
- the requirements should be mutually supportive,

- SOF claims ought to be consistent with the TOE objectives.

The first step – the Fig. 7.8 presents the TOE principal (functional) security requirements elaboration straight from the TOE security objectives analysis. Some security objectives have the proposed functional security requirements assigned (Fig. 7.4). First, each of them is analyzed, trying to merge the item expressed by the suggested component with the previously selected SFRs to obtain a more compact design. Usually, the merging takes place when the proposed item overlaps the previously selected items. When the merging is impossible, the proposed functional requirement can be attached to the set of previously selected principal SFRs as a separate item (joining). When the proposed item does not exist, the developer can freely choose CC functional components (expressed by the `FunComp` class object) or even define new ones (using CC component style) to cover the considered security objective.

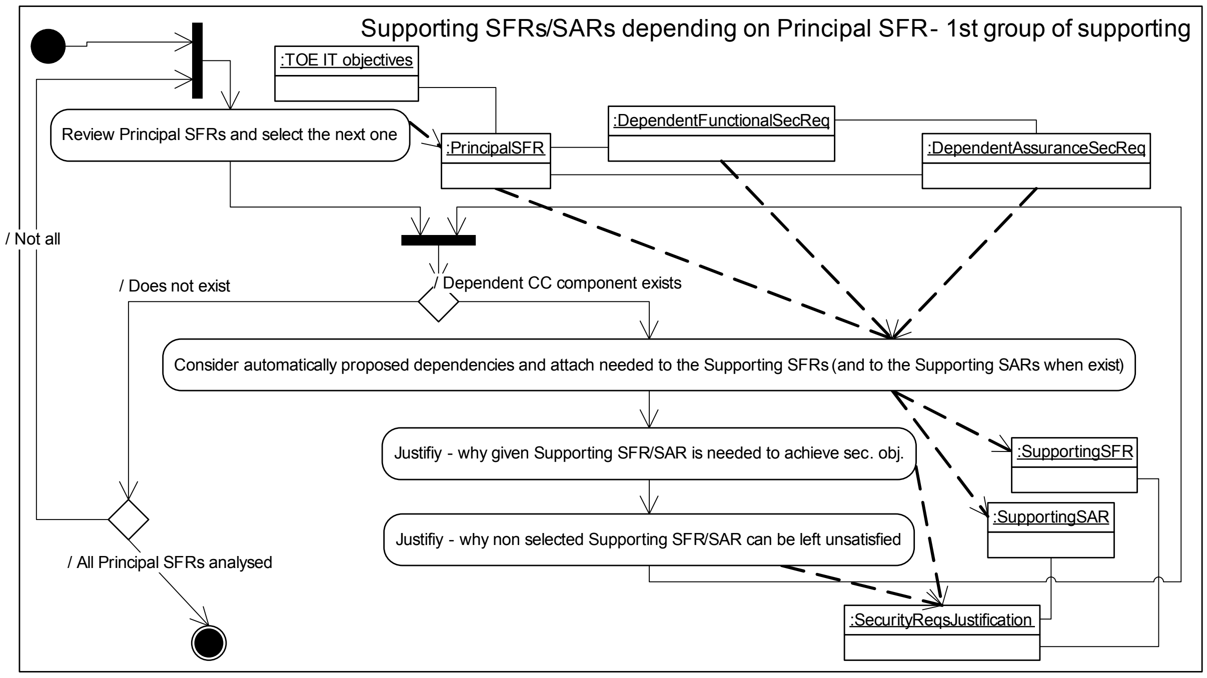

Supporting SFRs/SARs depending on Principal SFR- 1st group of supporting
:TOE IT objectives
Review Principal SFRs and select the next one
:PrincipalSFR
:DependentFunctionalSecReq
:DependentAssuranceSecReq
/ Not all
/ Does not exist
/ Dependent CC component exists
Consider automatically proposed dependences and attach needed to the Supporting SFRs (and to the Supporting SARs when exist)
Justifiy - why given Supporting SFR/SAR is needed to achieve sec. obj.
:SupportingSFR
:SupportingSAR
Justifiy - why non selected Supporting SFR/SAR can be left unsatisfied
/ All Principal SFRs analysed
:SecurityReqsJustification

Fig. 7.9. TOE functional supporting security requirements elaboration – the principal depending selection (CC-defined)

Rys. 7.9. Wypracowanie wspomagających, funkcjonalnych wymagań bezpieczeństwa dla przedmiotu oceny (TOE) – selekcja wymagań zależnych od podstawowych (zdefiniowanych przez standard)

Each of the components selected as principal is marked for future dependency analysis which is shown in the Fig. 7.9 below. Most of the CC catalogued components have their dependencies, in the form of a table defined by the standard. Depending components (functional or assurance) reinforce main components.

Each of the previously selected principal (functional-type) components can have its dependencies assigned by default, but not all of them are suitable in all circumstances. Some must be included into the design, some can be left unsatisfied, but all such cases need justification – a short comment why they are needed or can be rejected. The included dependencies are the first group of supporting requirements. Please note that they can be SFRs- or SARs-type.

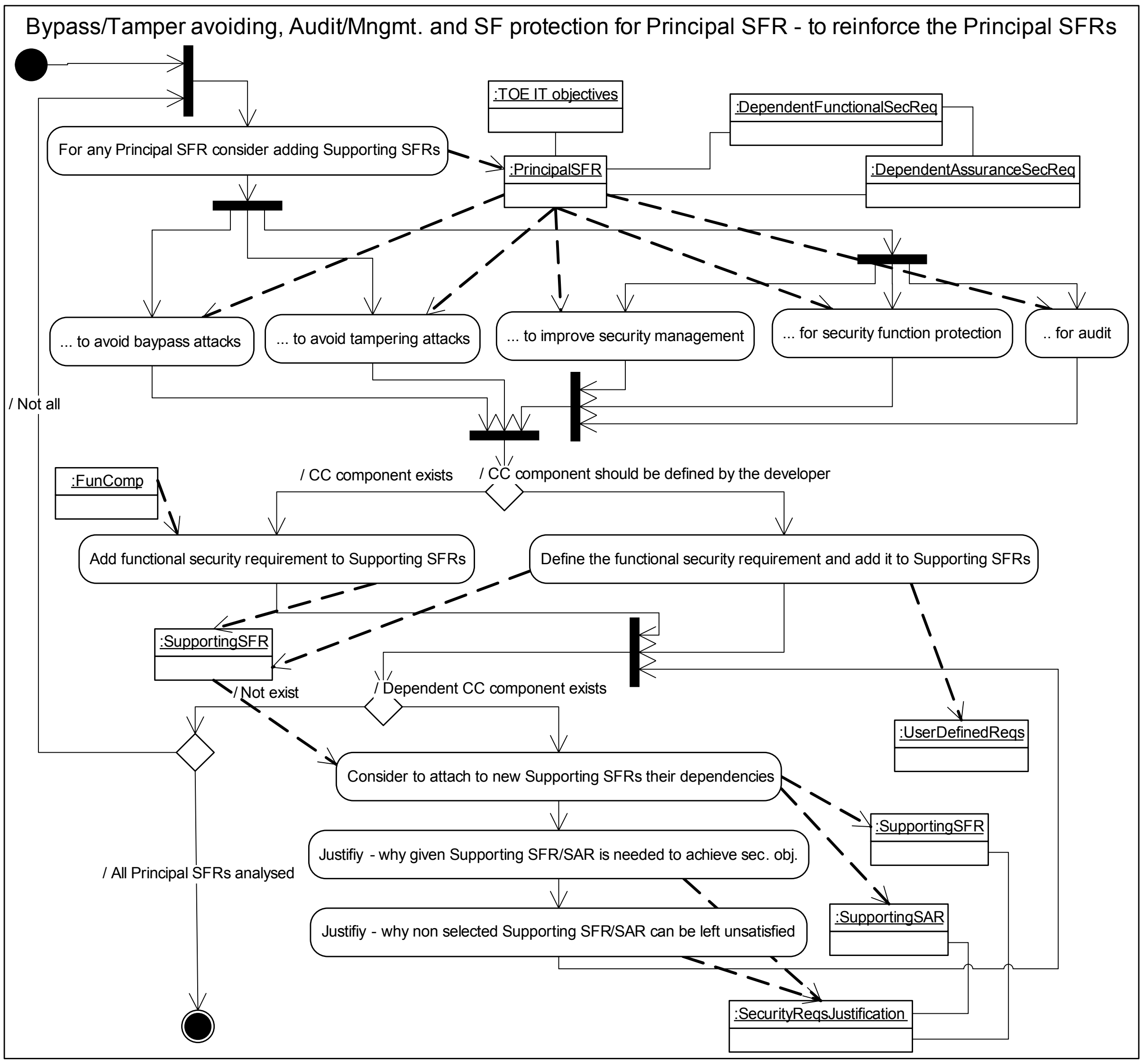

Bypass/Tamper avoiding, Audit/Mngmt. and SF protection for Principal SFR - to reinforce the Principal SFRs
:TOE IT objectives
:DependentFunctionalSecReq
For any Principal SFR consider adding Supporting SFRs
:PrincipalSFR
:DependentAssuranceSecReq
... to avoid baypass attacks
... to avoid tampering attacks
... to improve security management
... for security function protection
.. for audit
/ Not all
:FunComp
/ CC component exists
/ CC component should be defined by the developer
Add functional security requirement to Supporting SFRs
Define the functional security requirement and add it to Supporting SFRs
:SupportingSFR
/ Not exist
/ Dependent CC component exists
:UserDefinedReqs
Consider to attach to new Supporting SFRs their dependencies
:SupportingSFR
Justifiy - why given Supporting SFR/SAR is needed to achieve sec. obj.
/ All Principal SFRs analysed
:SupportingSAR
Justifiy - why non selected Supporting SFR/SAR can be left unsatisfied
:SecurityReqsJustification

Fig. 7.10. TOE functional supporting security requirements elaboration – the principal reinforcing selection

Rys. 7.10. Wypracowanie wspomagających, funkcjonalnych wymagań bezpieczeństwa dla przedmiotu oceny (TOE) – selekcja wymagań wzmacniających

After analyzing the dependencies of the principal SFRs and selecting the first group of supportive SFRs, the other selection process is recommended because the existing principal and supporting SFRs may not be enough to satisfy given security objectives. The Fig. 7.10 presents the selection process of the second group of supporting functional requirements.

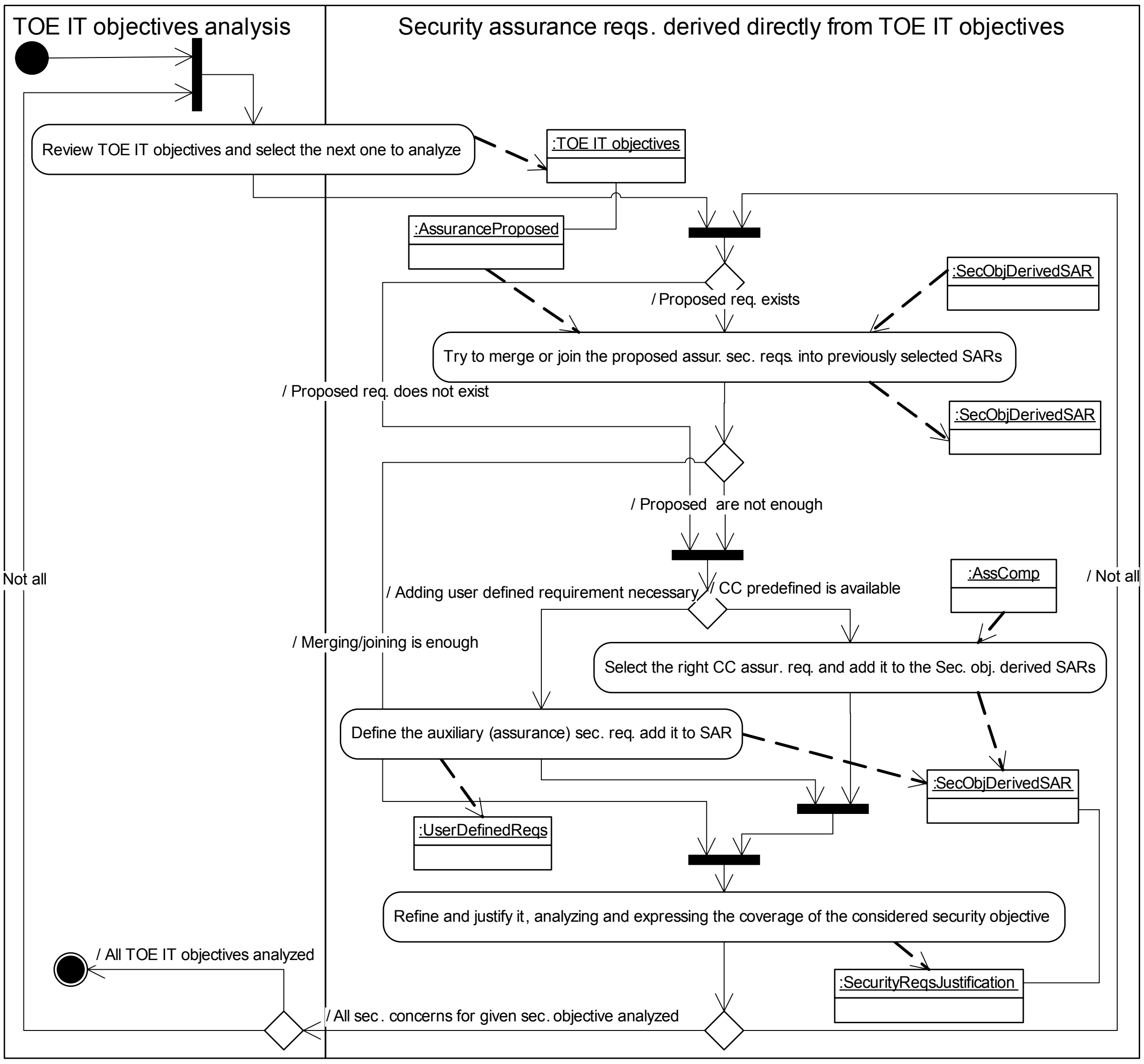

TOE IT objectives analysis
Security assurance reqs. derived directly from TOE IT objectives
Review TOE IT objectives and select the next one to analyze
:TOE IT objectives
:AssuranceProposed
:SecObjDerivedSAR
/ Proposed req. exists
Try to merge or join the proposed assur. sec. reqs. into previously selected SARs
/ Proposed req. does not exist
:SecObjDerivedSAR
/ Proposed are not enough
/ Not all
/ Adding user defined requirement necessary
/ CC predefined is available
:AssComp
/ Not all
/ Merging/joining is enough
Select the right CC assur. req. and add it to the Sec. obj. derived SARs
Define the auxiliary (assurance) sec. req. add it to SAR
:SecObjDerivedSAR
:UserDefinedReqs
Refine and justify it, analyzing and expressing the coverage of the considered security objective
/ All TOE IT objectives analyzed
:SecurityReqsJustification
/ All sec. concerns for given sec. objective analyzed

Fig. 7.11. TOE assurance security requirements elaboration – requirements derived directly from the security objectives
Rys. 7.11. Wypracowanie wymagań bezpieczeństwa uzasadniających zaufanie dla przedmiotu oceny (TOE) – selekcja wymagań wynikających z celów zabezpieczeń

The functional supporting security requirements are very important to achieve mutual support of requirements that should be demonstrated during the rationale process. Each of them must reinforce the principal selected one to better meet security objectives. The analysis is focused mainly on bypass or tampering attacks prevention, and reinforcement of the TOE security functions implied by the requirements. At this stage, the audit capabilities and the security management facilities are considered too. All needed, newly selected, supportive requirements and their dependencies of both types can be added to the supporting SFRs.

The next stage is the assurance security requirements elaboration which is as complicated as the functional security requirements elaboration process, discussed above. Please note that some assurance requirements were identified during the functional security requirements elaboration. These were SARs dependent on the principal SFRs. The next step is to identify all assurance requirements derived directly from the security objectives (Fig. 7.11). The process is very similar to the principal SFRs selection, presented in the Fig. 7.8, but the assurance security requirements shown in the Fig. 7.5 are used as the proposed components.

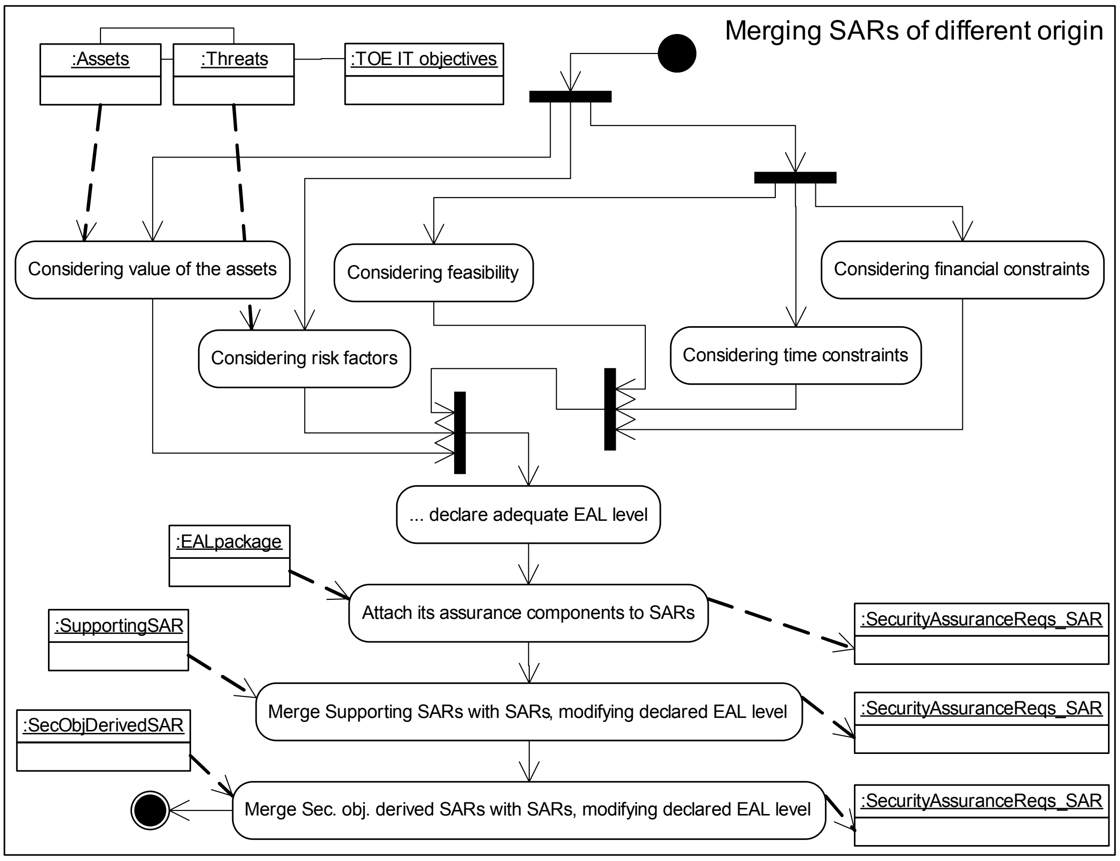

Merging SARs of different origin
:Assets
:Threats
:TOE IT objectives
Considering value of the assets
Considering feasibility
Considering financial constraints
Considering risk factors
Considering time constraints
... declare adequate EAL level
:EALpackage
Attach its assurance components to SARs
:SecurityAssuranceReqs_SAR
:SupportingSAR
Merge Supporting SARs with SARs, modifying declared EAL level
:SecurityAssuranceReqs_SAR
:SecObjDerivedSAR
Merge Sec. obj. derived SARs with SARs, modifying declared EAL level
:SecurityAssuranceReqs_SAR

Fig. 7.12. TOE assurance security requirements elaboration – merging different SARs with those influenced by declared EAL
Rys. 7.12. Wypracowanie wymagań bezpieczeństwa uzasadniających zaufanie dla przedmiotu oceny (TOE) – łączenie wymagań typu SAR, pochodzących z różnych źródeł, z wymaganiami zadeklarowanego poziomu uzasadnionego zaufania (EAL)

The CC developers favour the EAL declaring and use ready-made sets of corresponding assurance components. The Fig. 7.12 presents what factors can be considered while declaring EAL and how to merge different SARs with those coming from EALs. Please note that the risk value determined during the risk analysis has the key meaning. Please note that the SARs elaboration method presented there is very general and encompasses all possible ways of the assurance components specification.

The last step of the TOE security requirements elaboration is the preparation of the SOF claims, which is shown in the Fig. 7.13. It is recommended for EAL>1 when probabilistic or permutation mechanisms are assumed, like passwords, key generation, encryption, hash function, etc. The SOF claims express resistance to attacks on these mechanisms using a three-level predefined measure.

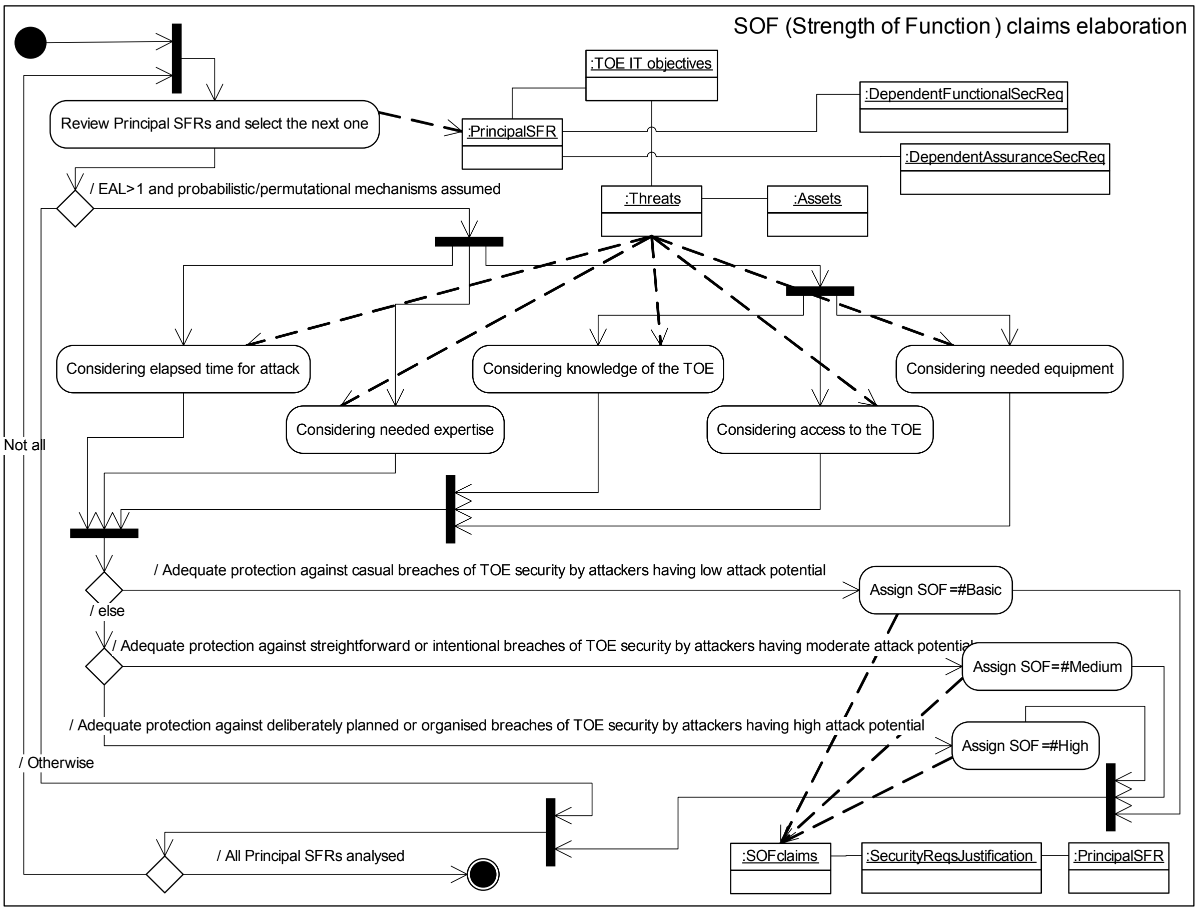

SOF (Strength of Function) claims elaboration
Review Principal SFRs and select the next one
:TOE IT objectives
:PrincipalSFR
:DependentFunctionalSecReq
:DependentAssuranceSecReq
:Threats
:Assets
/ EAL>1 and probabilistic/permutational mechanisms assumed
Considering elapsed time for attack
Considering needed expertise
Considering knowledge of the TOE
Considering access to the TOE
Considering needed equipment
/ Not all
/ Adequate protection against casual breaches of TOE security by attackers having low attack potential
Assign SOF=#Basic
/ else
/ Adequate protection against streightforward or intentional breaches of TOE security by attackers having moderate attack potential
Assign SOF=#Medium
/ Adequate protection against deliberately planned or organised breaches of TOE security by attackers having high attack potential
Assign SOF=#High
/ Otherwise
/ All Principal SFRs analysed
:SOFclaims
:SecurityReqsJustification
:PrincipalSFR

Fig. 7.13. SOF claims elaboration
Rys. 7.13. Wypracowanie deklaracji siły funkcji zabezpieczających (SOF)

The analysis is based mostly on the threats/risk analysis results and the factors taken into consideration are specified in the Fig. 7.13. The SOF claims should be consistent with the security objectives, based on the analysis of the attacker's capability. These claims are the supplement for the TOE security requirements.

7.3.2. TOE environment security requirements

The general elaboration scheme of the TOE environment security requirements, as a UML activity diagram, was shown in the Fig. 7.14. This is a refinement of the seventh step of the security requirements workout, shown above in the Fig. 7.7. Generally, the IT and non-IT factors should be considered. There are four main steps to be distinguished:

- working out the IT environment functional security requirements derived directly from the environment IT security objectives (the analogy to the principal SFRs for the TOE) – if the nature of the objectives allows for this, e.g. they are "well defined",

- joining their dependencies as supporting functional (or assurance) security requirements for the environment (see Fig. 7.2) – due to CC components dependencies, all of them should be analyzed, and then accepted or excluded,

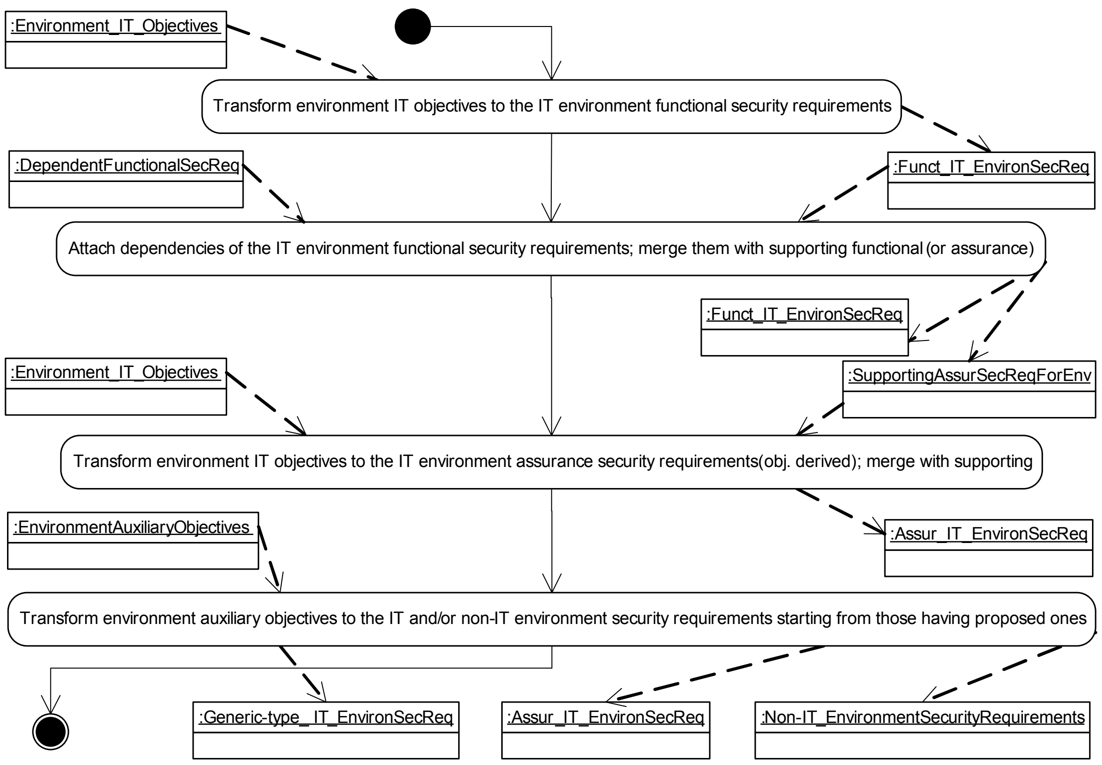

:Environment_IT_Objectives
Transform environment IT objectives to the IT environment functional security requirements
:DependentFunctionalSecReq
:Funct_IT_EnvironSecReq
Attach dependencies of the IT environment functional security requirements; merge them with supporting functional (or assurance)
:Funct_IT_EnvironSecReq
:Environment_IT_Objectives
:SupportingAssurSecReqForEnv
Transform environment IT objectives to the IT environment assurance security requirements(obj. derived); merge with supporting
:EnvironmentAuxiliaryObjectives
:Assur_IT_EnvironSecReq
Transform environment auxiliary objectives to the IT and/or non-IT environment security requirements starting from those having proposed ones
:Generic-type_ IT_EnvironSecReq
:Assur_IT_EnvironSecReq
:Non-IT_EnvironmentSecurityRequirements

Fig. 7.14. Environment security requirements elaboration – the main activity diagram
Rys. 7.14. Wypracowanie wymagań bezpieczeństwa dotyczących środowiska przedmiotu oceny (TOE) – nadrzędny diagram czynności

- identifying IT environment assurance requirements derived directly from the environment IT security objectives – if the nature of the objectives allows for this, e.g. they are "well defined",
- transforming the environment auxiliary objectives (see Fig. 6.12) to the IT and/or non-IT environment security requirements (see Fig. 7.6) starting from those having the proposed ones; usually, environment auxiliary objectives expressed on a more general detail level can successfully describe environmental security needs; they will be finally transformed to more general RE-family generics, sufficient to specify security requirements for the environment, without using CC components; please note that it is possible to add some components from the ACM (configuration management), AMA (assurance maintenance), AGD (guidance documentation), and ALC (life cycle support) CC classes [40].

The approach presented there is general and encompasses all possible ways of environment requirements specification. Each of the four specified stages will be shown on a separate activity diagram.

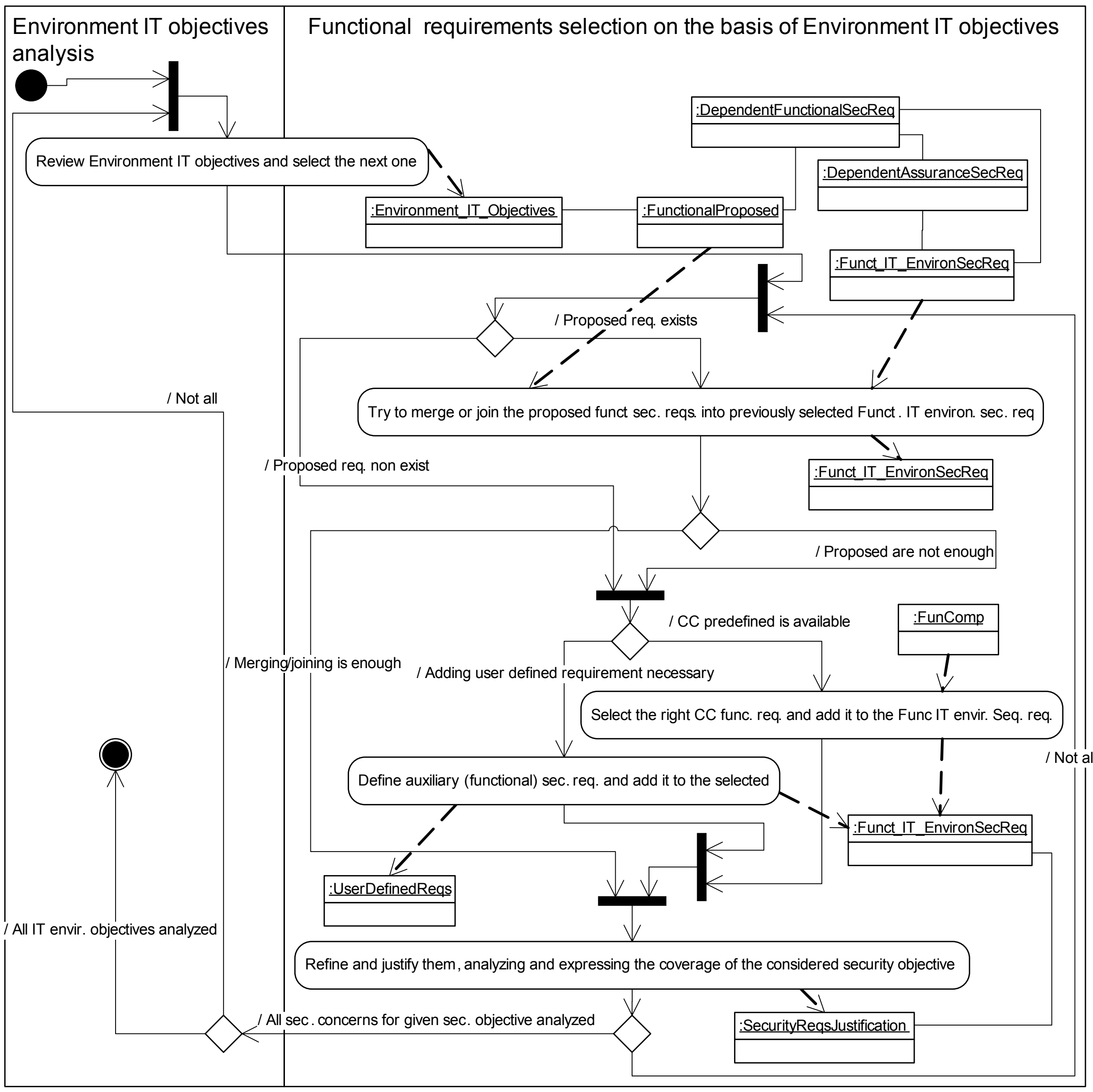

Environment IT objectives analysis
Functional requirements selection on the basis of Environment IT objectives
Review Environment IT objectives and select the next one
:DependentFunctionalSecReq
:DependentAssuranceSecReq
:Environment_IT_Objectives
:FunctionalProposed
:Funct_IT_EnvironSecReq
/ Proposed req. exists
/ Not all
Try to merge or join the proposed funct sec. reqs. into previously selected Funct. IT environ. sec. req
/ Proposed req. non exist
:Funct_IT_EnvironSecReq
/ Proposed are not enough
/ CC predefined is available
:FunComp
/ Merging/joining is enough
/ Adding user defined requirement necessary
Select the right CC func. req. and add it to the Func IT envir. Seq. req.
/ Not all
Define auxiliary (functional) sec. req. and add it to the selected
:Funct_IT_EnvironSecReq
:UserDefinedReqs
/ All IT envir. objectives analyzed
Refine and justify them, analyzing and expressing the coverage of the considered security objective
/ All sec. concerns for given sec. objective analyzed
:SecurityReqsJustification

Fig. 7.15. Functional IT environment security requirements derived from the environment IT security objectives
Rys. 7.15. Wypracowanie funkcjonalnych wymagań bezpieczeństwa dla środowiska informatycznego na podstawie celów zabezpieczeń o charakterze informatycznym

The Fig. 7.15 presents the elaboration of the functional IT environment security requirements derived straight from the environment IT security objectives (Fig. 7.14 - 1st step). This process is very similar to the TOE principal SFRs workout. The same types of the security objectives and requirements are used. Now they concern IT environment and will not be transformed to any security functions passed to the evaluation process. The functional security requirements, as a set of the selected CC components, are used to describe precisely the IT aspects of the environment when it is suitable and possible. Also in this case a dependency analysis for the selected components should be necessary.

The Fig. 7.16 presents the analysis of the functional IT environment security requirements (Fig. 7.14 – 2nd step) with respect to dependent components defined by the standard (functional or assurance). All of them are analyzed, some are selected for use, some are rejected. Any kind of decision needs justification.

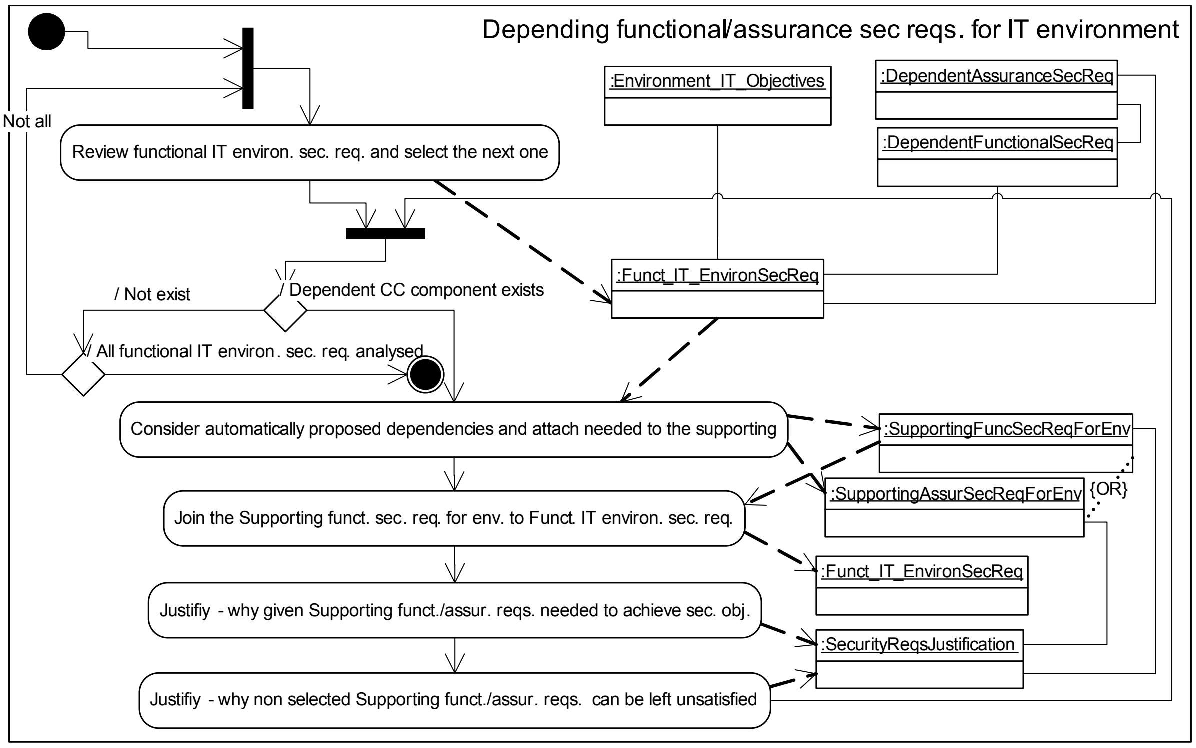

Depending functional/assurance sec reqs. for IT environment
/ Not all
Review functional IT environ. sec. req. and select the next one
/ Not exist
/ Dependent CC component exists
/ All functional IT environ. sec. req. analysed
Consider automatically proposed dependencies and attach needed to the supporting
Join the Supporting funct. sec. req. for env. to Funct. IT environ. sec. req.
Justifiy - why given Supporting funct./assur. reqs. needed to achieve sec. obj.
Justifiy - why non selected Supporting funct./assur. reqs. can be left unsatisfied
:Environment_IT_Objectives
:DependentAssuranceSecReq
:DependentFunctionalSecReq
:Funct_IT_EnvironSecReq
:SupportingFuncSecReqForEnv
:SupportingAssurSecReqForEnv
{OR}
:Funct_IT_EnvironSecReq
:SecurityReqsJustification

Fig. 7.16. Functional IT environment security requirements – identifying CC dependencies (functional and assurance)

Rys. 7.16. Wypracowanie funkcjonalnych wymagań bezpieczeństwa dla środowiska indormatycznego – identyfikacja wymagań zależnych, określonych w standardzie (funkcjonalych i uzasadniających zaufanie)

By the analogy to the TOE supporting SFRs and SARs, two kinds of requirements were introduced:

- Supporting functional security requirements for environment, joined/merged with the previously selected functional requirements,
- Supporting assurance security requirements for environment, joined/merged later (see Fig. 7.17), after identification of the assurance requirements derived directly from the environment IT objectives.

The Fig. 7.17 presents the workout of the assurance IT environment security requirements derived directly from the environment IT security objectives (Fig. 7.14 – 3rd step). This is very similar to the functional requirements selection (Fig. 7.15). It may be needed, though rarely.

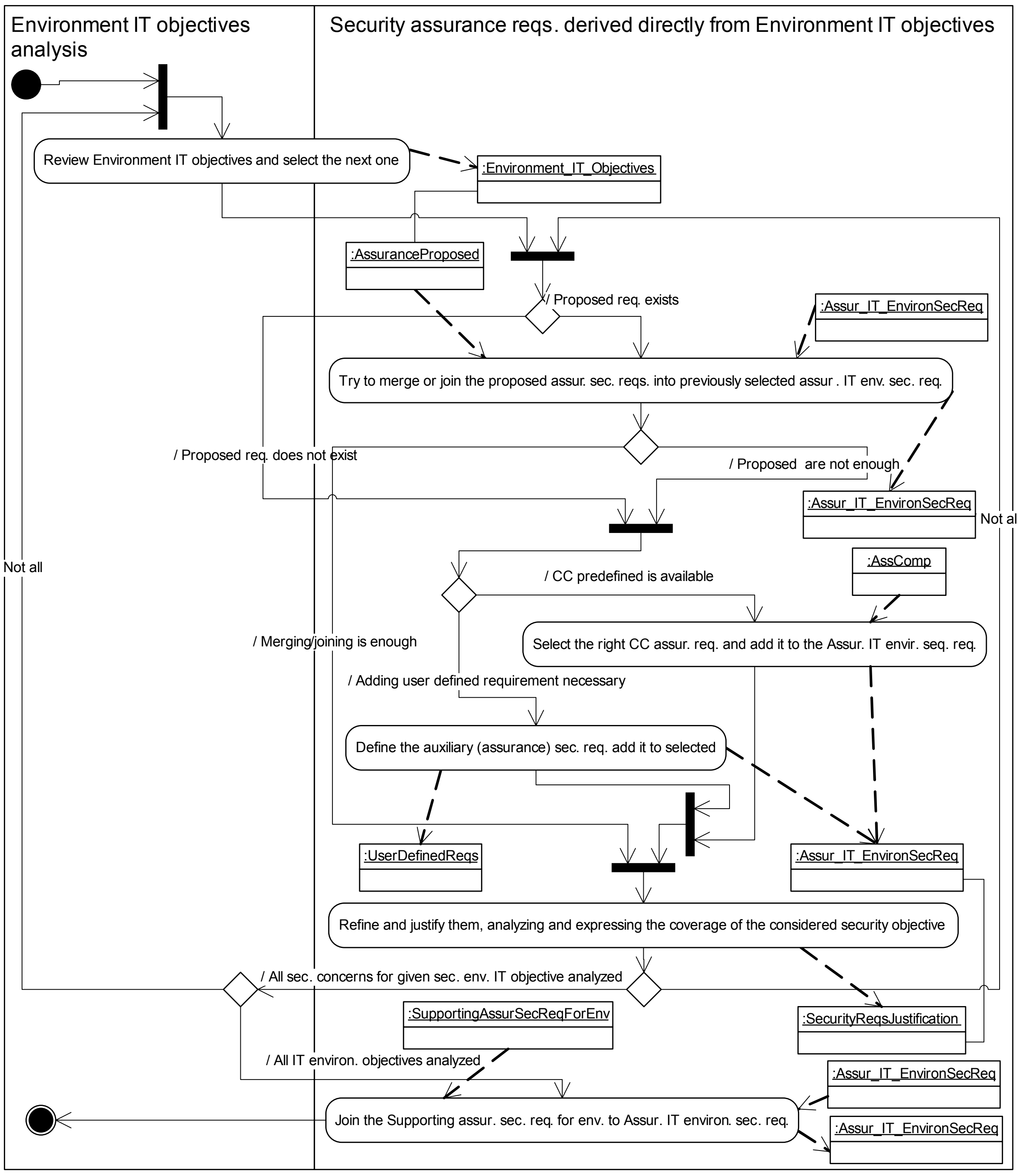

Environment IT objectives analysis
Security assurance reqs. derived directly from Environment IT objectives
Review Environment IT objectives and select the next one
:Environment_IT_Objectives
:AssuranceProposed
/ Proposed req. exists
:Assur_IT_EnvironSecReq
Try to merge or join the proposed assur. sec. reqs. into previously selected assur . IT env. sec. req.
/ Proposed req. does not exist
/ Proposed are not enough
:Assur_IT_EnvironSecReq
Not all
/ Not all
/ CC predefined is available
:AssComp
/ Merging/joining is enough
Select the right CC assur. req. and add it to the Assur. IT envir. seq. req.
/ Adding user defined requirement necessary
Define the auxiliary (assurance) sec. req. add it to selected
:UserDefinedReqs
:Assur_IT_EnvironSecReq
Refine and justify them, analyzing and expressing the coverage of the considered security objective
/ All sec. concerns for given sec. env. IT objective analyzed
:SupportingAssurSecReqForEnv
:SecurityReqsJustification
/ All IT environ. objectives analyzed
:Assur_IT_EnvironSecReq
Join the Supporting assur. sec. req. for env. to Assur. IT environ. sec. req.
:Assur_IT_EnvironSecReq

Fig. 7.17. Assurance IT environment security requirements derived from the environment IT security objectives – identifying and merging with others
Rys. 7.17. Wypracowanie wymagań bezpieczeństwa uzasadniających zaufanie dla środowiska informatycznego – ich identyfikacja i połączenie z określonymi poprzednio wymaganiami

The proposed assurance components are shown in the Fig. 7.5. They are the same as those used for TOE SARs, but they will not be evaluated and will be considered as dogmas during the evaluation.

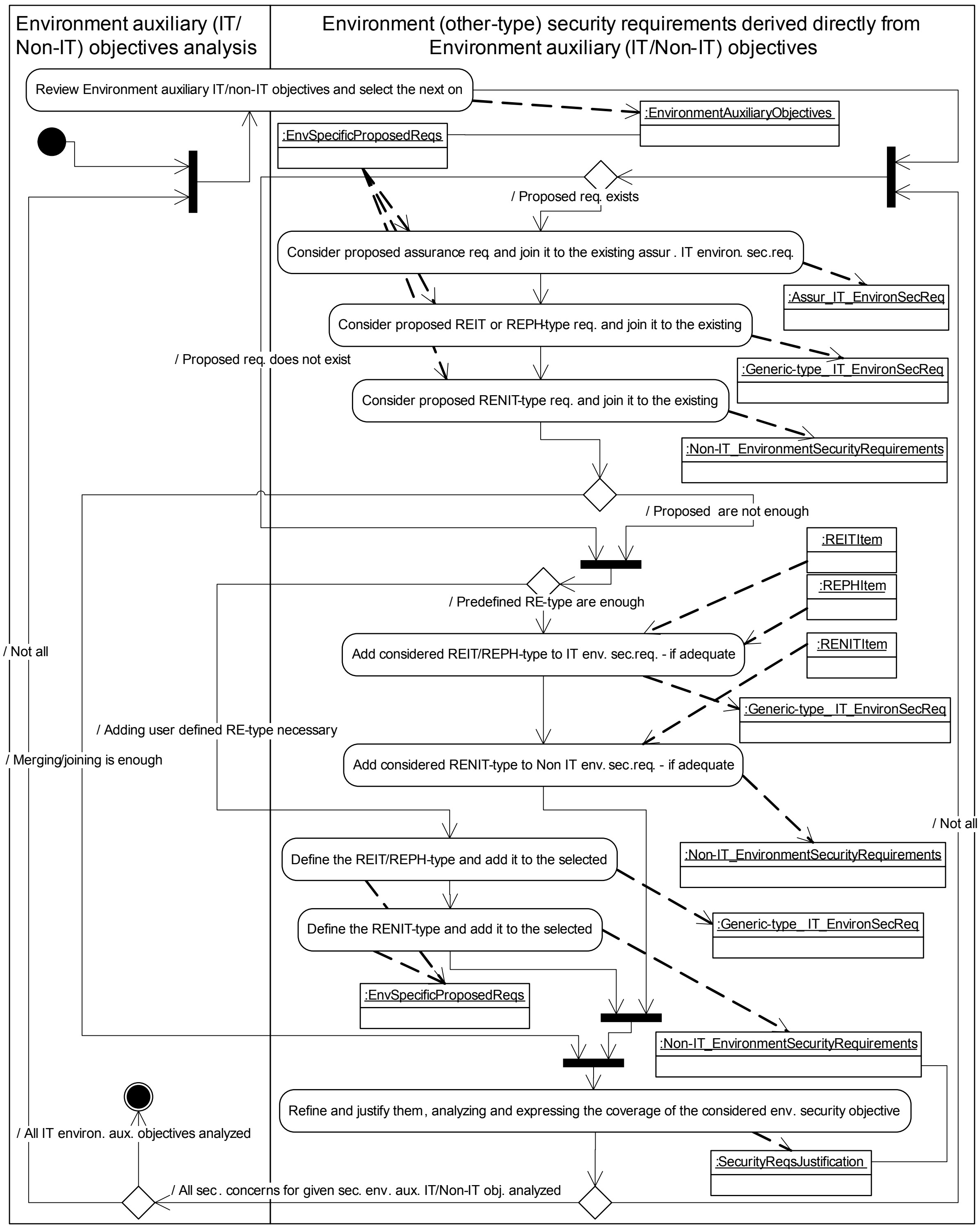

Environment auxiliary (IT/ Non-IT) objectives analysis
Environment (other-type) security requirements derived directly from Environment auxiliary (IT/Non-IT) objectives
Review Environment auxiliary IT/non-IT objectives and select the next on
:EnvironmentAuxiliaryObjectives
:EnvSpecificProposedReqs
/ Proposed req. exists
Consider proposed assurance req. and join it to the existing assur. IT environ. sec.req.
:Assur_IT_EnvironSecReq
Consider proposed REIT or REPH-type req. and join it to the existing
:Generic-type_IT_EnvironSecReq
/ Proposed req. does not exist
Consider proposed RENIT-type req. and join it to the existing
:Non-IT_EnvironmentSecurityRequirements
/ Proposed are not enough
:REITItem
:REPHItem
/ Predefined RE-type are enough
Add considered REIT/REPH-type to IT env. sec.req. - if adequate
:RENITItem
/ Not all
:Generic-type_IT_EnvironSecReq
/ Adding user defined RE-type necessary
/ Merging/joining is enough
Add considered RENIT-type to Non IT env. sec.req. - if adequate
/ Not all
:Non-IT_EnvironmentSecurityRequirements
Define the REIT/REPH-type and add it to the selected
:Generic-type_IT_EnvironSecReq
Define the RENIT-type and add it to the selected
:EnvSpecificProposedReqs
:Non-IT_EnvironmentSecurityRequirements
Refine and justify them, analyzing and expressing the coverage of the considered env. security objective
/ All IT environ. aux. objectives analyzed
:SecurityReqsJustification
/ All sec. concerns for given sec. env. aux. IT/Non-IT obj. analyzed

Fig. 7.18. Auxiliary, IT/non-IT environment security requirements finishing
Rys. 7.18. Dokończenie specyfikacji wymagań dla środowiska z użyciem wymagań pomocniczych, informatycznych i poza informatycznych

Finally, the identified requirements derived from the security objectives are merged with the supportive ones to the Assurance IT environment security requirements.

The above mentioned IT environment requirements specification manners are suitable when IT environment objectives are precise enough and can be transformed to CC components (see Fig. 7.4 and Fig. 7.5). Very often a more general approach can be enough. For this reason three families of generics were defined:

- OEIT-/REIT-families to express IT environment aspects,
- OEPH-/REPH-families to get a broader view of this environment and to consider its technical (physical) aspects,
- OSMN-/RENIT-families to encompass all organizational and human aspects.

The Fig. 7.18 (Fig. 7.14 – 4th step) presents the elaboration process based on the environment auxiliary security objectives (Fig. 7.6). The result was the issue of both environment IT and/or non-IT requirements. Please note that some assurance components cannot be excluded. All RE-family requirements are bound and the REIT- or REPH- families are reinforced by the RENIT-family.

All assigned requirements should be justified to facilitate the future rationale process. The completed set of requirements is the basis for implementing the security functions of the TOE.

7.4. Formal approach to the security requirements specification

The security requirements (Fig. 7.2) are much more complicated than the previous stages. They can be expressed in a simplified manner and the discussion concerns the final shape of the specification only. During the elaboration process many temporary specifications are created, which can be added later to the general model when needed.

Definition 7.1: Security requirements.

The security requirements are the triplet of sets:

$S_{SecReqs} = \langle S_{SecReqsItems}, S_{SecReqsPars}, S_{SecReqsAssocs} \rangle$.

1. The first element of the triplet, $S_{SecReqsItems}$ is the sum of three disjointed[16] sets of generics:

$S_{SecReqsItems} =$
$S_{TOE\_SecurityRequirements} \cup S_{IT\_EnvironmentSecurityRequirements} \cup S_{Non\text{-}IT\_EnvironmentSecurityRequirements}$ and
$S_{TOE\_SecurityRequirements} \cap S_{IT\_EnvironmentSecurityRequirements} \cap S_{Non\text{-}IT\_EnvironmentSecurityRequirements} = \varnothing$,
where:

$S_{TOE\_SecurityRequirements} = S_{SecurityFunctionalReqs\_SFR} \cup S_{SecurityAssuranceReqs\_SAR}$,
$S_{SecurityFunctionalReqs\_SFR} = \{\texttt{FunSecClass} | \texttt{FunSecClass}.\mathit{assignstat}\text{=\#ASSIGNED}\}$,
$S_{SecurityAssuranceReqs\_SAR} = \{\texttt{AssSecClass} | \texttt{AssSecClass}.\mathit{assignstat}\text{=\#ASSIGNED}\}$,
$S_{IT\_EnvironmentSecurityRequirements} =$
$\{\texttt{FunSecClass} \mid \texttt{FunSecClass}.\mathit{assignstat}\text{=\#ASSIGNED}\} \cup$
$\{\texttt{AssSecClass} \mid \texttt{AssSecClass}.\mathit{assignstat}\text{=\#ASSIGNED}\} \cup$

[16] It ought to be true when specification is finished.

$\{$`REgrGeneric` | `REgrGeneric`.*assignstat*=#ASSIGNED$\}$.

$S_{Non\text{-}IT\_EnvironmentSecurityRequirements}$ =

$\{$`REgrGeneric` | `REgrGeneric`.*assignstat*=#ASSIGNED$\}$.

2. The second element of the triplet, $S_{SecReqsPars}$ is the sum of two sets of parameterization associations:

$S_{SecReqsPars} \subseteq$ GenParAssoc and $S_{SecReqsPars} = \{$`ParamDA4RE`, `ParamS4RE`$\}$.

3. The final element of the triplet, $S_{SecReqsAssocs}$ is the sum of three sets of associations dealing with mapping the security requirements to the security objectives items:

$S_{SecReqsAssocs} \subseteq$ SecAssoc and $S_{SecReqsAssocs}$ =

$\{$`FunSec4Ogr`, `AssSec4Ogr`, `REgr4Ogr`$\}$.

□

The parameters concern the RE-group generic items only. Other component parameters – textual, enumeration, selected from available lists – are considered during the component refinement. Usually, generic-type parameters of the `REgrGeneric` class items are left uncompleted, though, when used, they ought to be properly assigned as others:

```
ParamDA4RE
REgrGeneric.paramDAgr=DAgrGeneric or DAgrGeneric->isEmpty
ParamS4RE
REgrGeneric.paramSgr=SgrGeneric or SgrGeneric->isEmpty
```

□

The general definition of this stage can be formulated as follows:

Definition 7.2: Security requirements elaboration.

The security requirements elaboration is the transformation of S_{SecObj} to $S_{SecReqs}$, i.e. transformation of $S_{SecObjItems}$ with their parameters $S_{SecObjPars}$, to $S_{SecReqsItems}$ with their parameters $S_{SecReqsPars}$, according to the mapping rules represented by the $S_{SecReqsAssocs}$ associations.

□

At this stage a very important element is the principal SFRs subset: $S_{PrincipalSFRs} \subseteq S_{SecReqsItem}$. The security functions are constructed around any single principal SFR or small group of principal SFRs.

8. Workout of the TOE summary specification (TSS)

8.1. TOE summary specification defined by Common Criteria

The standard says that the aim of the development of the TOE Summary Specification (TSS) is to specify the TOE solution and demonstrate how the TOE provides security functions and assurance measures to satisfy the defined TOE security requirements [38]. The TSS specification should include (Fig. 8.1):

- a definition of the IT security functions (SF), expressed by F-family generics, satisfying SFRs and SARs;
- a definition of assurance measures which satisfy the identified SARs.

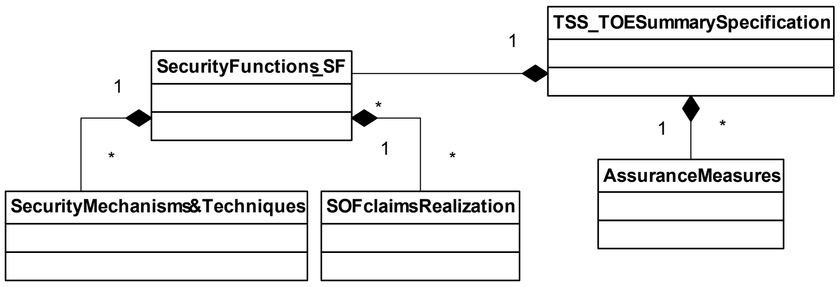


Fig. 8.1. TOE summary specification (TSS) for the ST – the CC defined basic elements
Rys. 8.1. Specyfikacja końcowa przedmiotu oceny dla zadania zabezpieczeń (TSS) – elementy określone w standardzie Wspólne kryteria

This description should:

- focus on what the TOE should provide to meet SFRs and SARs,
- be on "high level" – not providing implementation details,
- refer to the TOE design and testing documentation, administrators and user guides,
- be in conformance with the Common Criteria functional paradigm [39], [10],
- use TOE-specific terminology.

The TSS elaboration process is briefly and informally described in the standard. This chapter shows its semiformal modelling and implementation, assuring the consistency with other development stages encompassed by the IT Security Development Framework. The specification means for this stage are F-family generics and informal text. The concept included in the standard, expressed on a UML diagram (Fig. 8.1), will be the basis for the developed models.

8.2. TSS data model

The TSS is worked out on the basis of security requirements. The Fig. 8.2 presents more details dealing with the main classes participating in the elaboration of the TSS.

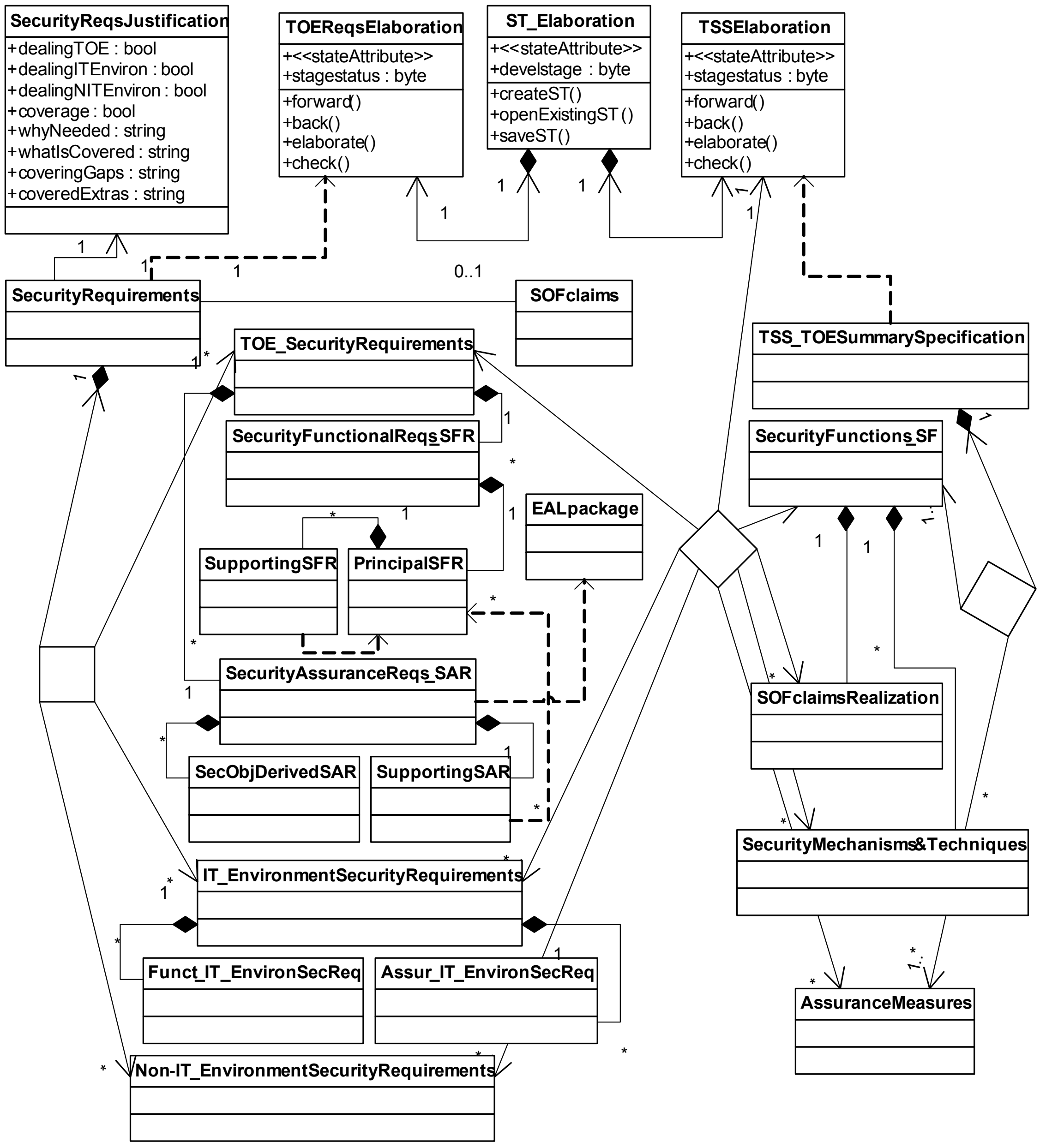


Fig. 8.2. The TSS elaboration (the right side) on the basis of the TOE security requirements (the left side) – the assumed static structure diagram

Rys. 8.2. Wypracowanie końcowej specyfikacji przedmiotu oceny (TSS) (strona prawa) na podstawie wymagań bezpieczeństwa TOE (strona lewa) – założona struktura statyczna

The left part of the figure deals with input data – showing all security requirements used to build the TSS specification. The `TSSElaboration` class responsible for the TSS elaboration is presented on the right.

The most important input data are Principal SFRs directly transferred to the SF. The supportive SFRs are considered during SF refinement. Additionally, it should be checked how the declared SOF claims will be satisfied by the SF. The proposed security functions concern the *`Fgr4FunSec`* security association class (Fig. 3.20) with *assocstat* = #PROPOSED (Fig. 3.17). In this case the proposed SFs are related to some functional requirements, selected as the Principal SFRs, expressing common security issues (functionality), e.g. login, encryption, hashing, digital signature verification, etc. Please note that F-family generics are expressed on a very general level of abstraction. They are used rather for reference purposes and as containers for high level specifications of security functions.

Assurance measures, usually represented by a reference list of documents, may be considered as evidence material delivered to evaluators.

8.3. TSS elaboration

To elaborate the TOE Summary Specification (TSS) it is recommended to follow the steps shown in the Fig. 8.3. This activity diagram presents the behaviour of the `TSSElaboration` class.

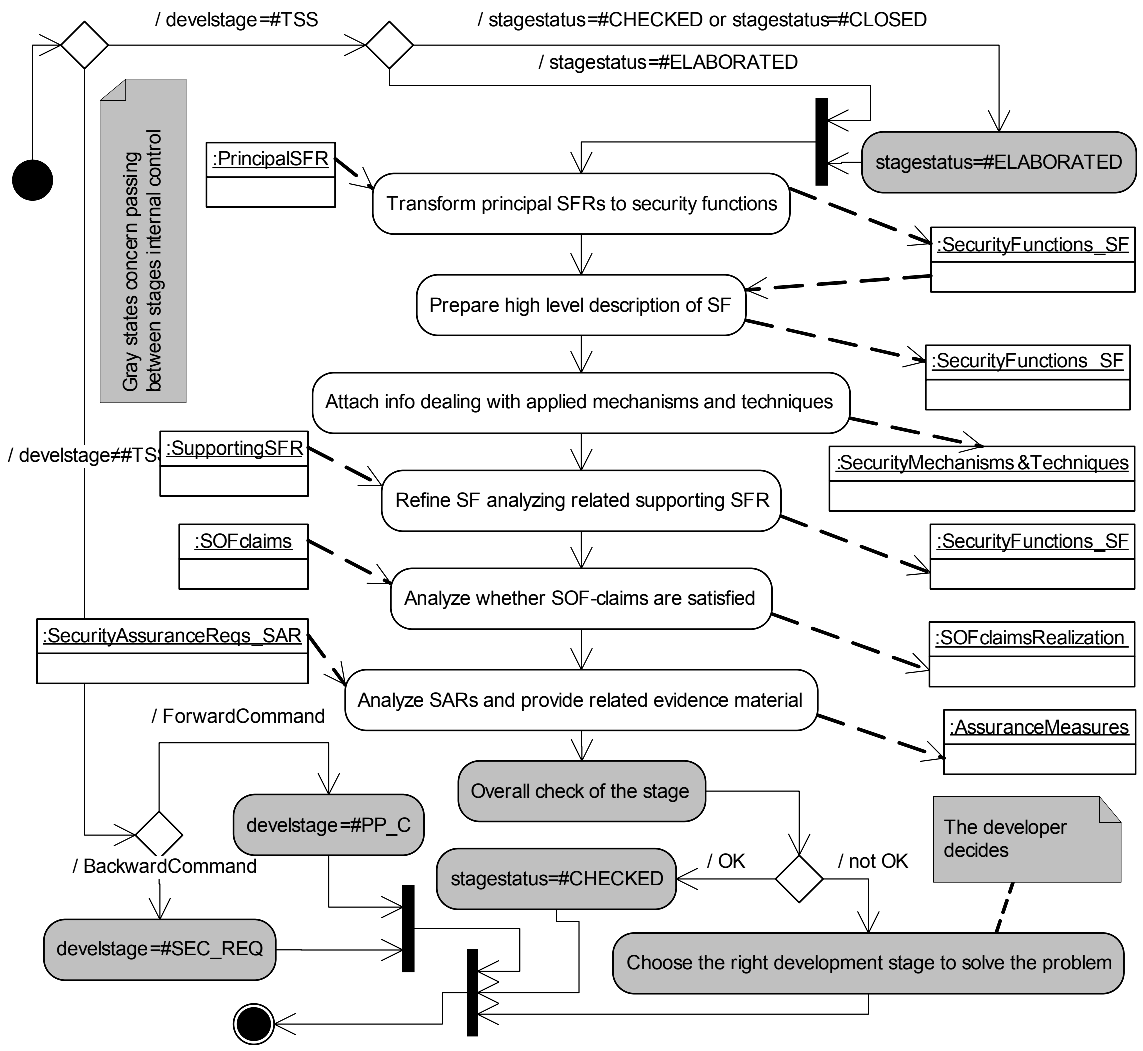
First, the set of the principal SFRs is analyzed. The SFRs are grouped according to security concerns. To each group or, if it is impossible, to a given requirement, a single function is assigned.

A high level description for any of the functions meeting principal SFRs is prepared and the reference list of related mechanisms and techniques implementing them is updated (optionally). It should be noted that mechanisms and techniques are: cryptographic algorithms, claims of conformance to standards, etc.

Then the supporting SFRs related to a given function are analyzed and, on this basis, the security function specification is refined.

Please note that during the TOE security functional requirements elaboration some SOF claims may be added (Fig. 7.13) and now they can be verified. It is considered whether security functions will satisfy SOF claims.

Any security function can be carried out on a different level of rigour but every time it is expressed by SARs. For every SAR the applied assurance measures are assigned in the form of a reference. The list is the evidence material ensuring that all SARs are satisfied.



/ develstage=#TSS
/ stagestatus=#CHECKED or stagestatus=#CLOSED
/ stagestatus=#ELABORATED
Gray states concern passing between stages internal control
stagestatus=#ELABORATED
:PrincipalSFR
Transform principal SFRs to security functions
:SecurityFunctions_SF
Prepare high level description of SF
:SecurityFunctions_SF
Attach info dealing with applied mechanisms and techniques
:SecurityMechanisms&Techniques
/ develstage≠#TSS
:SupportingSFR
Refine SF analyzing related supporting SFR
:SecurityFunctions_SF
:SOFclaims
Analyze whether SOF-claims are satisfied
:SOFclaimsRealization
:SecurityAssuranceReqs_SAR
Analyze SARs and provide related evidence material
:AssuranceMeasures
/ ForwardCommand
develstage=#PP_C
Overall check of the stage
The developer decides
/ BackwardCommand
stagestatus=#CHECKED
/ OK
/ not OK
develstage=#SEC_REQ
Choose the right development stage to solve the problem

Fig. 8.3. TOE summary specification for the ST elaboration – the main activity diagram
Rys. 8.3. Wypracowanie końcowej specyfikacji przedmiotu oceny (TSS) dla zadania zabezpieczeń – nadrzędny diagram czynności

The security requirements (both types) should be suitably met by the IT security functions and assurance measures. Taking this goal into consideration during TSS workout makes further TSS rationale easier.

8.4. Formal approach to the TOE summary specification

The TOE summary specification (Fig. 8.2) can also be expressed in a more formalized way. In this case formal mechanisms can be helpful especially to transform the security functional requirements, grouped around the principal SFRs, to the security functions (SF). The TSS can be considered as a pair of sets (generic-type parameters do not exist).

Definition 8.1: TOE summary specification.

The TOE summary specification is the pair of sets:

$S_{TSS} = \langle S_{TSS\_SecFuns}, S_{TSS\_SecAssocs} \rangle.$

1. The first element of the pair, S_{TSS_SecFun} is represented by the F group items, expressing the security functions:

$S_{TSS\_SecFuns} = \{\texttt{FgrGeneric} \mid \texttt{FgrGeneric}.assignstat\text{=\#ASSIGNED}\}$.

2. The second element of the pair, $S_{TSS\_SecAssocs}$ expresses associations dealing with mapping the security functions to the security requirements, exactly to the principal SFRs:

$S_{TSS\_SecAssocs} \subseteq \mathsf{SecAssoc}$ and $S_{TSS\_SecAssocs} = \{\texttt{Fgr4FunSec}\}$.

□

Definition 8.2: TOE summary specification elaboration.

The TOE summary specification elaboration is the transformation of $S_{SecReqs}$ to S_{TSS}, i.e. transformation of $S_{PrincipalSFRs} \subseteq S_{SecReqsItem}$, to $S_{TSS\_SecFuns}$, according to the mapping rules represented by the $S_{TSS\_SecAssocs}$ associations.

□

9. PROTECTION PROFILE CLAIMS

Although the protection profile claims are rather trivial in comparison with the other stages, they will be presented in a separate chapter to keep a unified approach to the presentation of the IT security development process. In this case only the data structure is shown. For ST specifications based on the existing Protection Profiles, PP claims should be specified (Fig. 9.1). PP claims include [38]:

- a reference identifying the PP to which compliance is claimed,
- refinements applied to the PP (PP tailoring),
- TOE additions satisfied by the ST, dealing with the security objectives and/or the security requirements.

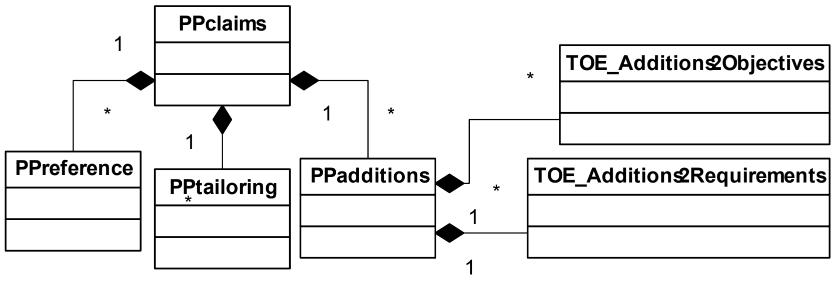


Fig. 9.1. PP claims for security target (ST) – the CC defined basic elements
Rys. 9.1. Deklaracje profili zabezpieczeń (PP) dla zadania zabezpieczeń (ST) – elementy określone w standardzie Wspólne kryteria

The `PreparePPclaims` class (Fig. 2.4) is responsible for this stage. Its behaviour will not be discussed here. The PP claims elements should be considered during the rationale too.

10. RATIONALE PROCESS

The standard says that at the end of the development process it should be demonstrated that the conformable TOE provided by its countermeasures will be secure in its environment [38]. The rationale can be expressed by two classes diagrams (Fig. 10.1 and Fig. 10.2).

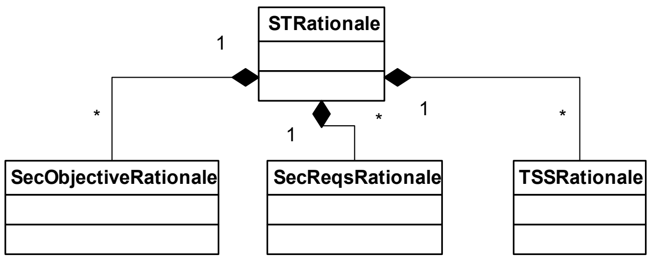


Fig. 10.1. ST rationale – the CC defined basic elements
Rys. 10.1. Uzasadnienie zadania zabezpieczeń (ST) – elementy określone w standardzie Wspólne kryteria

Please note that the ST rationale has a specific element – the TSS rationale is more complex than the PP rationale. For this reason, it will be discussed in details.

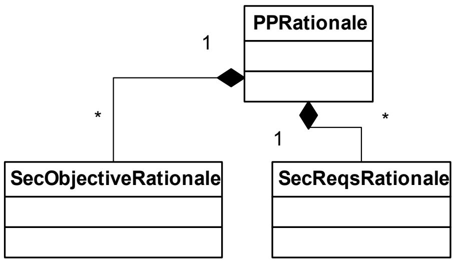


Fig. 10.2. PP rationale – the CC defined basic elements
Rys. 10.2. Uzasadnienie profilu zabezpieczeń (PP) – elementy określone w standardzie Wspólne kryteria

The standard also says that the IT security development stages rationales must be performed, showing that:

- all aspects of the identified security needs expressed by the TOE security environment are suitably addressed by the security objectives and the assumptions are upheld by them – it is called the security objectives rationale;

- the security objectives for the TOE are suitably met by the identified IT security requirements (SFR, SAR), the requirements are mutually supportive, and SOF claims are consistent with the TOE objectives – it is called the security requirements rationale;
- security requirements (SFR, SAR) are suitably met by the IT security functions and assurance measures (a list prepared by the developer); this rationale step deals with ST only – it is called the TSS rationale.

Common Criteria assume to perform and document the rationale at the end of the development process, and this may be difficult to carry out. The presented method offers some flexibility, allowing to reveal any gaps in the design earlier, after each design stage, and distinguishing two phases in each rationale stage:

- current short justification of any element when selected, putting "why it is needed to cover a given security aspect" into words,
- rationale, summarizing partial justifications, may be considered as "Common Criteria rationale".

The added current justification phase ensures more consistency, due to the obligatory designer's declaration, but coverage status helps control if any specification item is necessary and sufficient.

Most arguments used for the rationale process are sampled during previous stages of the elaboration, dealing with matters as they come. For that reason, now they can only be refined to be more consistent and adequate.

10.1. Security objectives rationale

The elaboration of the security objectives rationale (eight steps), i.e. the behaviour of the `SecObjectiveRationale` class (Fig. 10.1), is presented in the Fig. 10.3. This class works on the same data structures as the `TOESecObjElaboration` class does, providing the optimization of its contents.

During the first step all redundant objectives should be revealed. Those unnecessary can be removed. Some objectives can be merged with others or properly related to the security environment elements (Fig. 10.4 is a refinement of the first activity shown in the Fig. 10.3). It is checked if each security objective covers at least one threat, OSP or assumption (redundancy checking – all left must be necessary). Also the security environment elements, like threats, OSPs or assumptions should be checked if each of them is covered by at least one security objective (Fig. 10.5 is a refinement of the second activity shown in the Fig. 10.3).

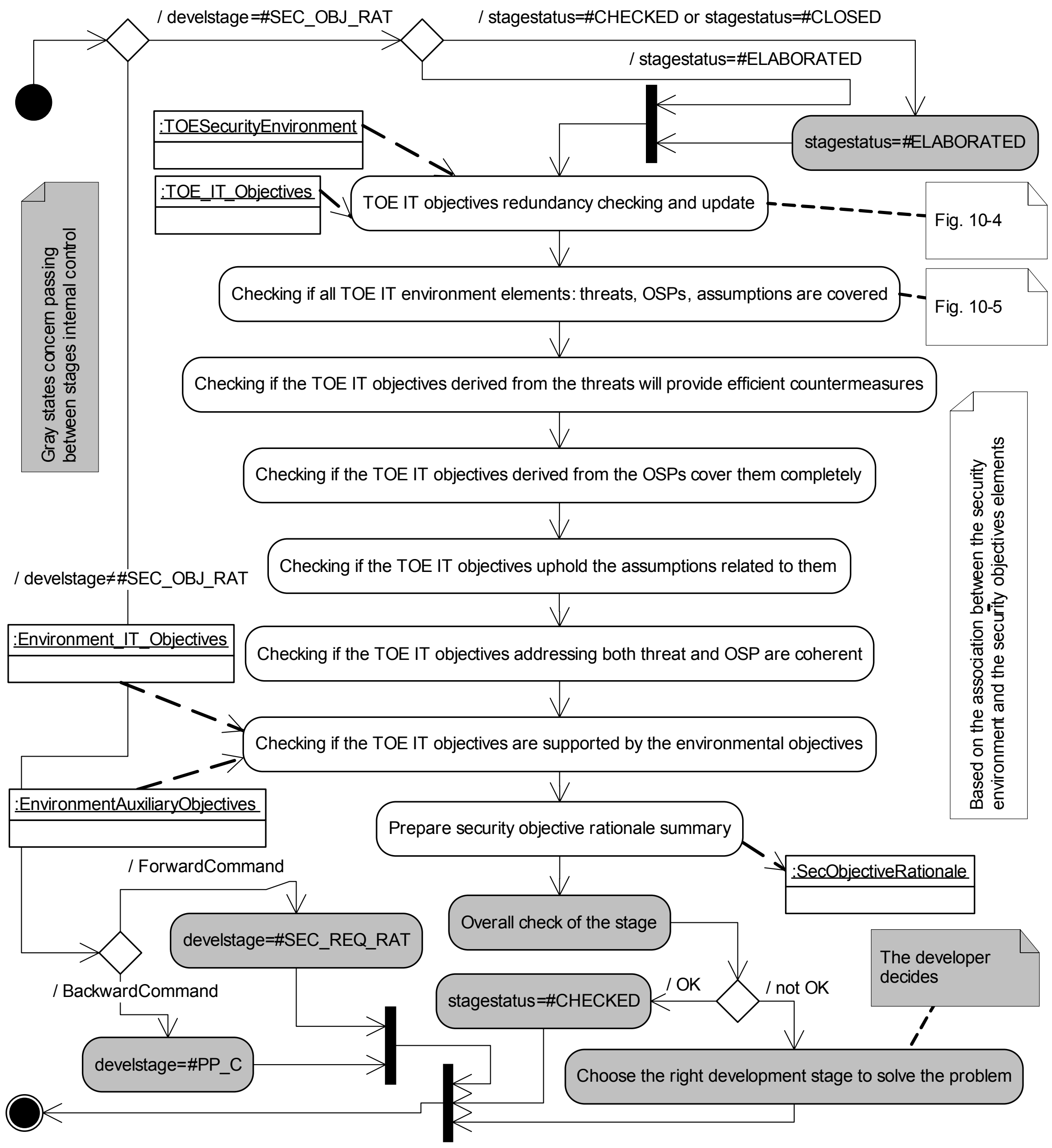

/ develstage=#SEC_OBJ_RAT
/ stagestatus=#CHECKED or stagestatus=#CLOSED
/ stagestatus=#ELABORATED
stagestatus=#ELABORATED
:TOESecurityEnvironment
:TOE_IT_Objectives
TOE IT objectives redundancy checking and update
Fig. 10-4
Checking if all TOE IT environment elements: threats, OSPs, assumptions are covered
Fig. 10-5
Gray states concern passing between stages internal control
Checking if the TOE IT objectives derived from the threats will provide efficient countermeasures
Checking if the TOE IT objectives derived from the OSPs cover them completely
Based on the association between the security environment and the security objectives elements
/ develstage≠#SEC_OBJ_RAT
Checking if the TOE IT objectives uphold the assumptions related to them
:Environment_IT_Objectives
Checking if the TOE IT objectives addressing both threat and OSP are coherent
Checking if the TOE IT objectives are supported by the environmental objectives
:EnvironmentAuxiliaryObjectives
Prepare security objective rationale summary
:SecObjectiveRationale
/ ForwardCommand
develstage=#SEC_REQ_RAT
Overall check of the stage
The developer decides
/ BackwardCommand
stagestatus=#CHECKED
/ OK
/ not OK
develstage=#PP_C
Choose the right development stage to solve the problem

Fig. 10.3. The security objectives rationale – the general activity diagram
Rys. 10.3. Uzasadnienie celów zabezpieczeń – nadrzędny diagram czynności

These two steps optimize the association between the security environment and the objectives, allowing to complete this analysis. The security objectives must be sufficient. Apart from the two refined activities, the following actions are performed:

- checking if any security objective addressed to the threat will provide efficient countermeasures (detective, preventing, corrective) to it – a simple built-in risk analyzer may be useful;
- considering if security objectives dealing with OSPs will provide their complete coverage;

- considering if security objectives will provide the upholding of the assumptions related to them;
- considering the role of those objectives which address both the threat and OSP;
- considering the supportive role of any environmental objectives for the TOE objectives; this activity, rather trivial and not refined there, looks like those shown in the Fig.10.3: for each TOE IT objective it is necessary to review environment IT objectives and auxiliary objectives related to them, modifying the existing or adding new ones to better support the analyzed TOE IT objective.

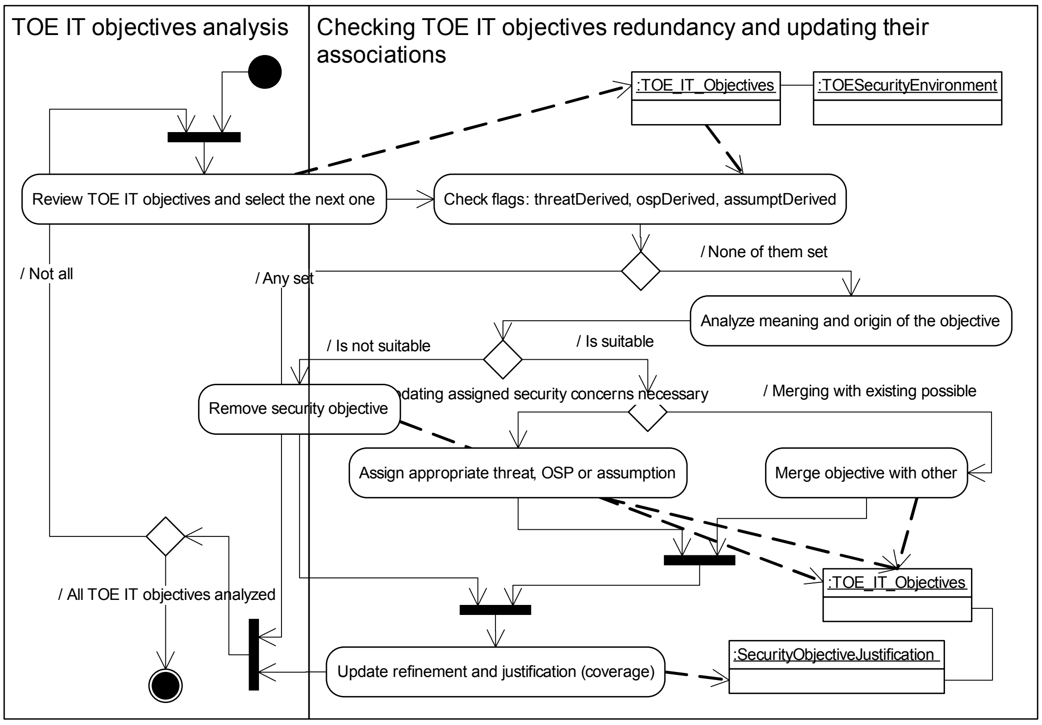
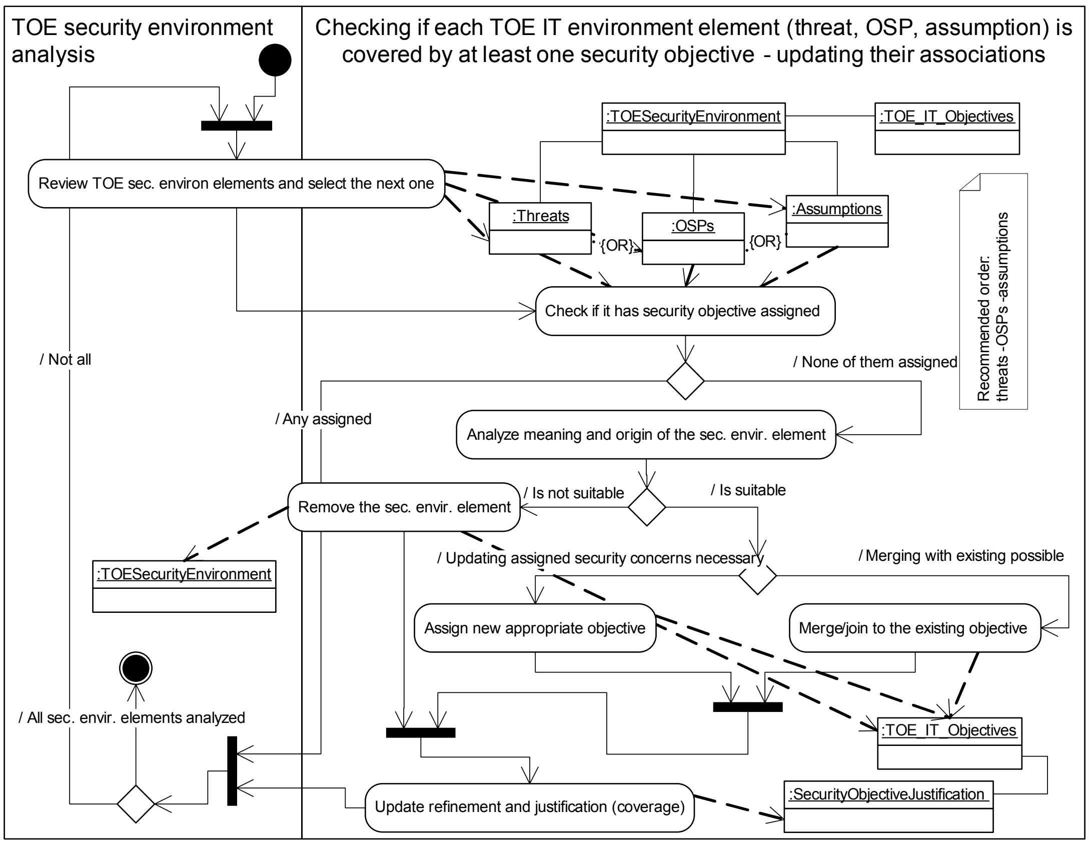

TOE IT objectives analysis
Checking TOE IT objectives redundancy and updating their associations
:TOE_IT_Objectives
:TOESecurityEnvironment
Review TOE IT objectives and select the next one
Check flags: threatDerived, ospDerived, assumptDerived
/ None of them set
/ Not all
/ Any set
Analyze meaning and origin of the objective
/ Is not suitable
/ Is suitable
Remove security objective
Updating assigned security concerns necessary
/ Merging with existing possible
Assign appropriate threat, OSP or assumption
Merge objective with other
:TOE_IT_Objectives
/ All TOE IT objectives analyzed
:SecurityObjectiveJustification
Update refinement and justification (coverage)

Fig. 10.4. The security objectives specification redundancy checking
Rys. 10.4. Wykazanie braku nadmiarowości specyfkacji celów zabezpieczeń



TOE security environment analysis
Checking if each TOE IT environment element (threat, OSP, assumption) is covered by at least one security objective - updating their associations
:TOESecurityEnvironment
:TOE_IT_Objectives
Review TOE sec. environ elements and select the next one
:Threats
:OSPs
:Assumptions
{OR}
{OR}
Recommended order: threats -OSPs -assumptions
Check if it has security objective assigned
/ Not all
/ None of them assigned
/ Any assigned
Analyze meaning and origin of the sec. envir. element
Remove the sec. envir. element
/ Is not suitable
/ Is suitable
:TOESecurityEnvironment
/ Updating assigned security concerns necessary
/ Merging with existing possible
Assign new appropriate objective
Merge/join to the existing objective
:TOE_IT_Objectives
/ All sec. envir. elements analyzed
Update refinement and justification (coverage)
:SecurityObjectiveJustification

Fig. 10.5. The security environment specification redundancy checking
Rys. 10.5. Wykazanie braku nadmiarowości specyfkacji otoczenia zabezpieczeń

10.2. Security requirements rationale

The successful completion of the security objectives rationale allows to go to the second, more restricted stage – the security requirements rationale. The security requirements rationale encompasses the activities presented in the Fig. 10.6. It expresses the behaviour of the `SecReqsRationale` class (Fig. 10.1). Additionally, this class is based on the data structures used by the `TOEReqsElaboration` class. The activities (Fig. 10.6), refined in a few further figures, concern the following checkings:

- if SFRs are suitable (should be necessary and sufficient, each of the identified SFRs is sufficient to satisfy a given security objective, how environmental objectives support TOE objectives);
- if SARs are appropriate for the TOE (sufficient, not excessive to address security objectives and technically feasible to the TOE);
- if SOF claims are appropriate (consistency with security objectives, based on the analysis of the attacker's capability);

- if security requirements are mutually supportive (SFRs and SARs dependencies satisfaction, internal consistency, avoidance of bypassing or tampering attacks).

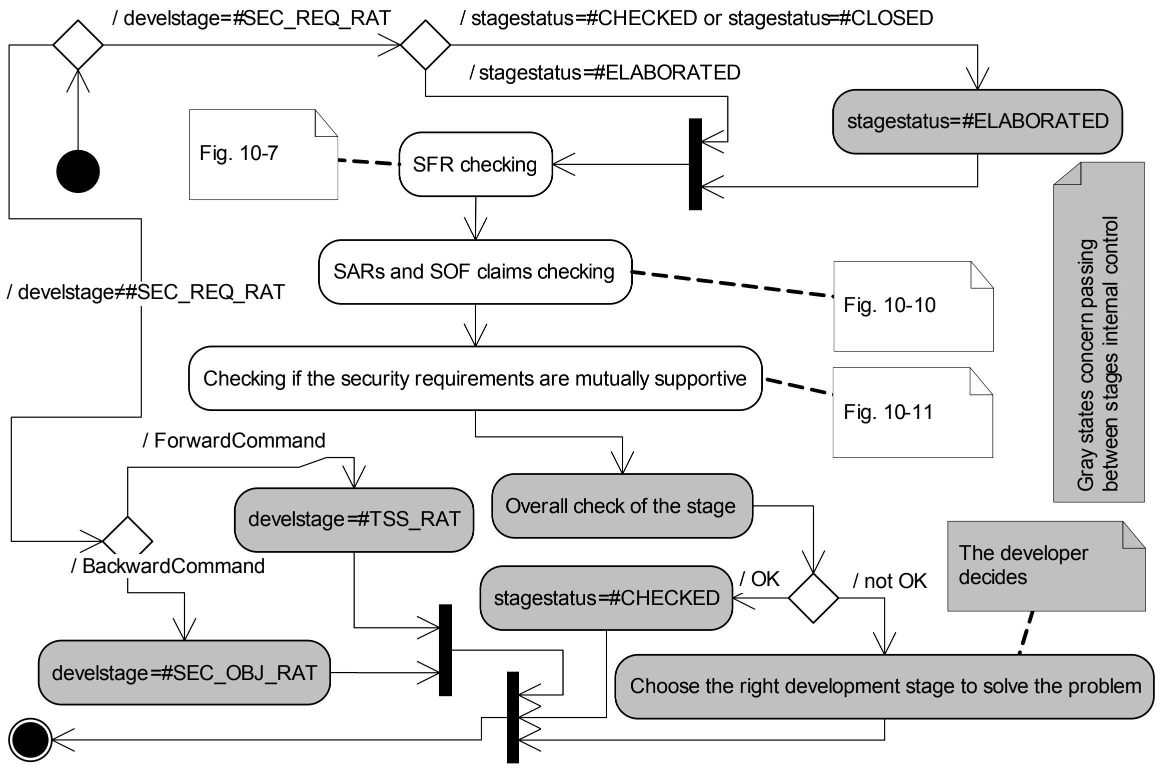

/ develstage=#SEC_REQ_RAT
/ stagestatus=#CHECKED or stagestatus=#CLOSED
/ stagestatus=#ELABORATED
stagestatus=#ELABORATED
Fig. 10-7
SFR checking
SARs and SOF claims checking
/ develstage≠#SEC_REQ_RAT
Fig. 10-10
Checking if the security requirements are mutually supportive
Fig. 10-11
Gray states concern passing between stages internal control
/ ForwardCommand
develstage=#TSS_RAT
Overall check of the stage
/ BackwardCommand
stagestatus=#CHECKED
/ OK
/ not OK
The developer decides
develstage=#SEC_OBJ_RAT
Choose the right development stage to solve the problem

Fig. 10.6. The security requirements rationale – the general activity diagram
Rys. 10.6. Uzasadnienie wymagań bezpieczeństwa przedmiotu oceny – nadrzędny diagram czynności

The elaboration of the security requirements rationale begins from SFRs checking presented in the Fig. 10.7.

During the rationale process it should be demonstrated that the IT security requirements (particularly the SFRs) are necessary and sufficient. For this reason, the review of the relationships between them and the security objectives is performed.

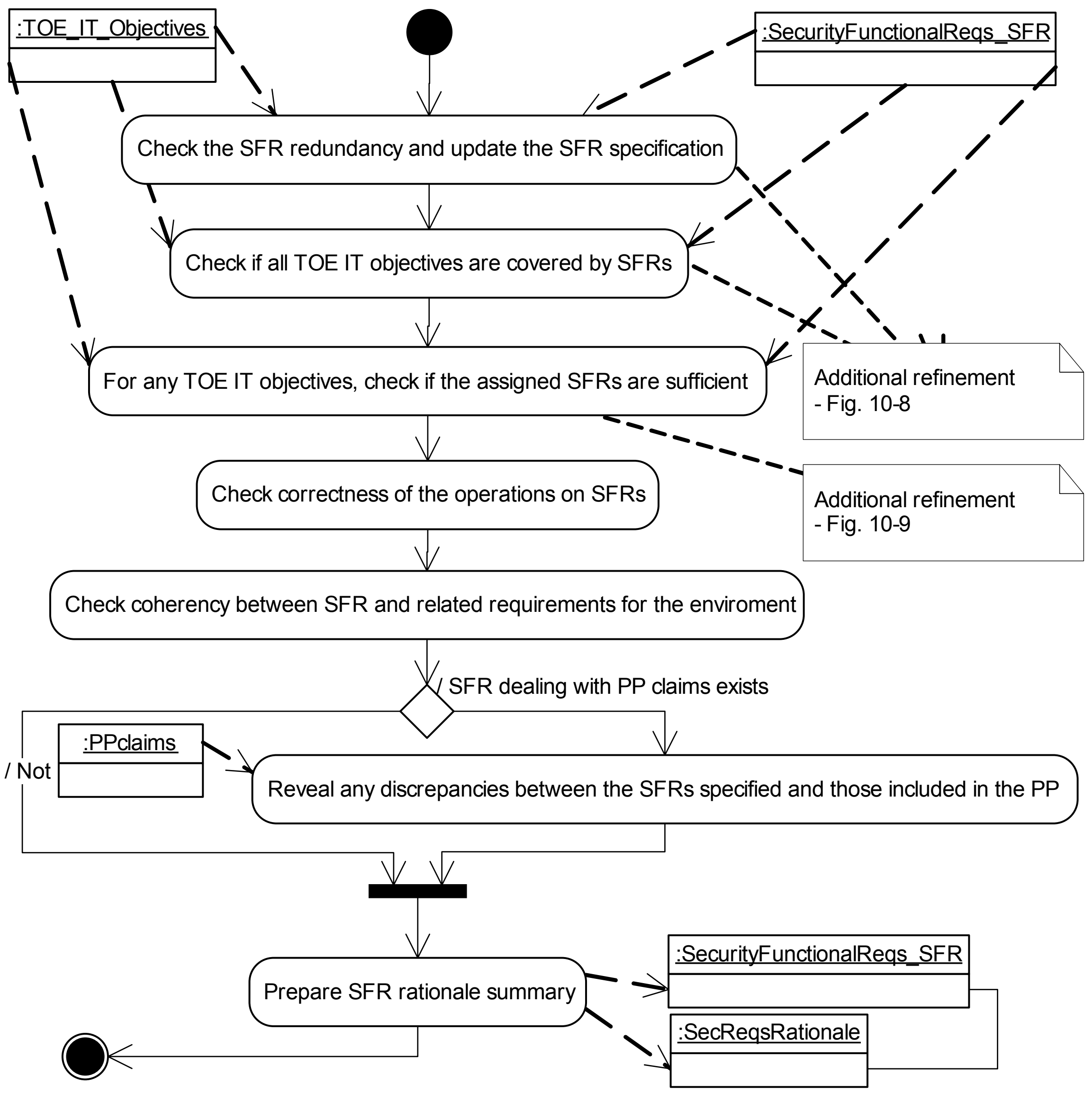

:TOE_IT_Objectives
:SecurityFunctionalReqs_SFR
Check the SFR redundancy and update the SFR specification
Check if all TOE IT objectives are covered by SFRs
For any TOE IT objectives, check if the assigned SFRs are sufficient
Additional refinement - Fig. 10-8
Additional refinement - Fig. 10-9
Check correctness of the operations on SFRs
Check coherency between SFR and related requirements for the enviroment
/ SFR dealing with PP claims exists
/ Not
:PPclaims
Reveal any discrepancies between the SFRs specified and those included in the PP
Prepare SFR rationale summary
:SecurityFunctionalReqs_SFR
:SecReqsRationale

Fig. 10.7. The security requirements rationale – SFRs checking
Rys. 10.7. Uzasadnienie wymagań bezpieczeństwa przedmiotu oceny – sprawdzenie funkcjonalnych wymagań bezpieczeństwa (SFR)

First, all redundant SFRs should be revealed and then removed or properly assigned (Fig. 10.8 is a refinement of the first two activities shown in the Fig. 10.7).

Additionally, the security objectives that are not properly assigned are revealed (Fig. 10.9 is a refinement of the third activity shown in the Fig. 10.7).

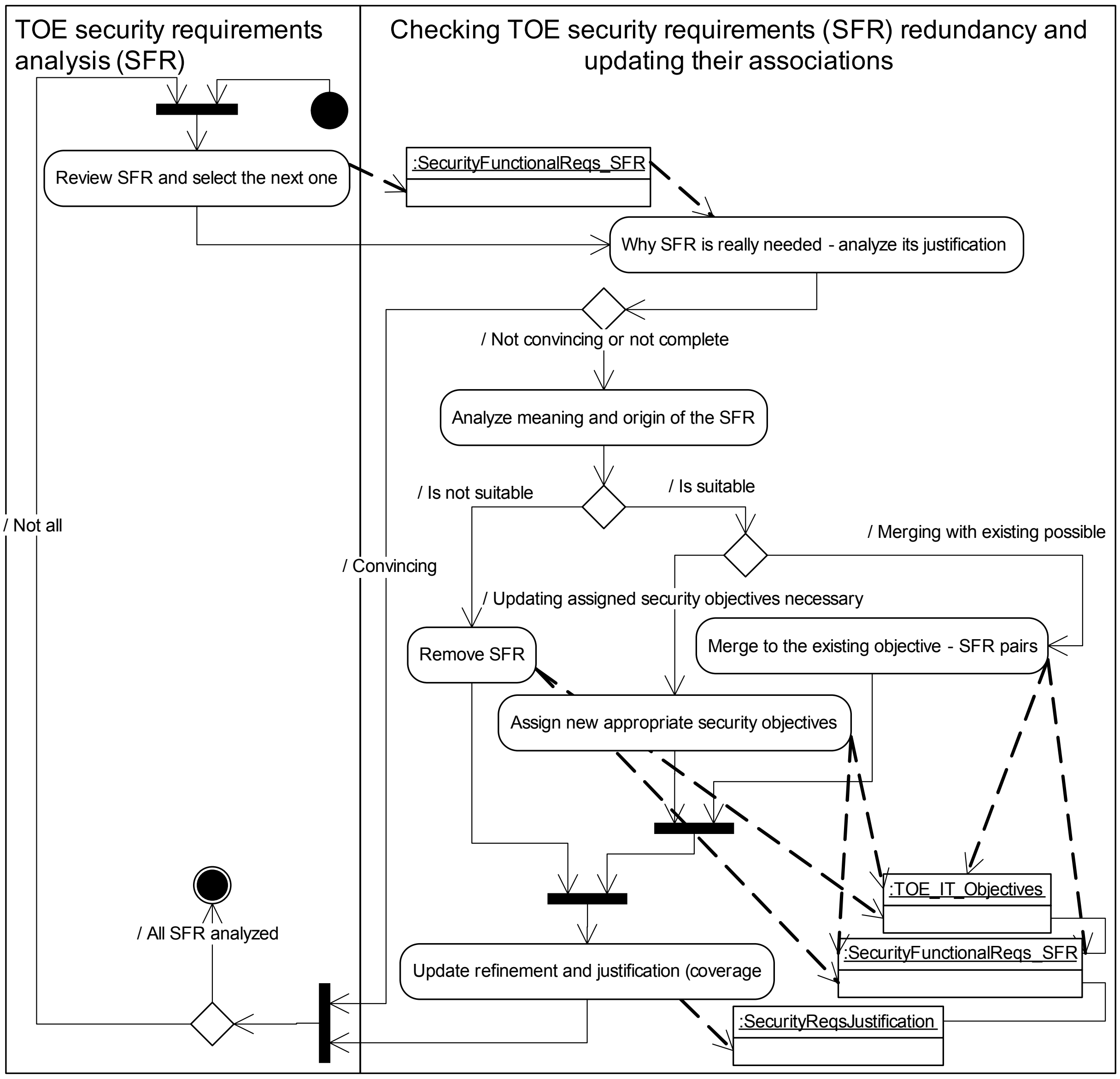

TOE security requirements analysis (SFR)
Checking TOE security requirements (SFR) redundancy and updating their associations
Review SFR and select the next one
:SecurityFunctionalReqs_SFR
Why SFR is really needed - analyze its justification
/ Not convincing or not complete
Analyze meaning and origin of the SFR
/ Is not suitable
/ Is suitable
/ Merging with existing possible
/ Not all
/ Convincing
/ Updating assigned security objectives necessary
Remove SFR
Merge to the existing objective - SFR pairs
Assign new appropriate security objectives
:TOE_IT_Objectives
:SecurityFunctionalReqs_SFR
/ All SFR analyzed
Update refinement and justification (coverage
:SecurityReqsJustification

Fig. 10.8. The security requirements rationale – revealing redundant SFRs
Rys. 10.8. Uzasadnienie wymagań bezpieczeństwa przedmiotu oceny – identyfikacja nadmiarowych funkcjonalnych wymagań bezpieczeństwa (SFR)

The association of security objectives and SFRs, including previously defined classes, is updated, allowing to provide further analysis.

The next step of the functional security requirements rationale (Fig. 10.7) is checking the correctness of operations on SFRs and the SFRs compliance with the environment objectives. When some of the security requirements are specified within the claimed PP, they must be checked for the compliance too. Finally, the rationale summary is prepared.

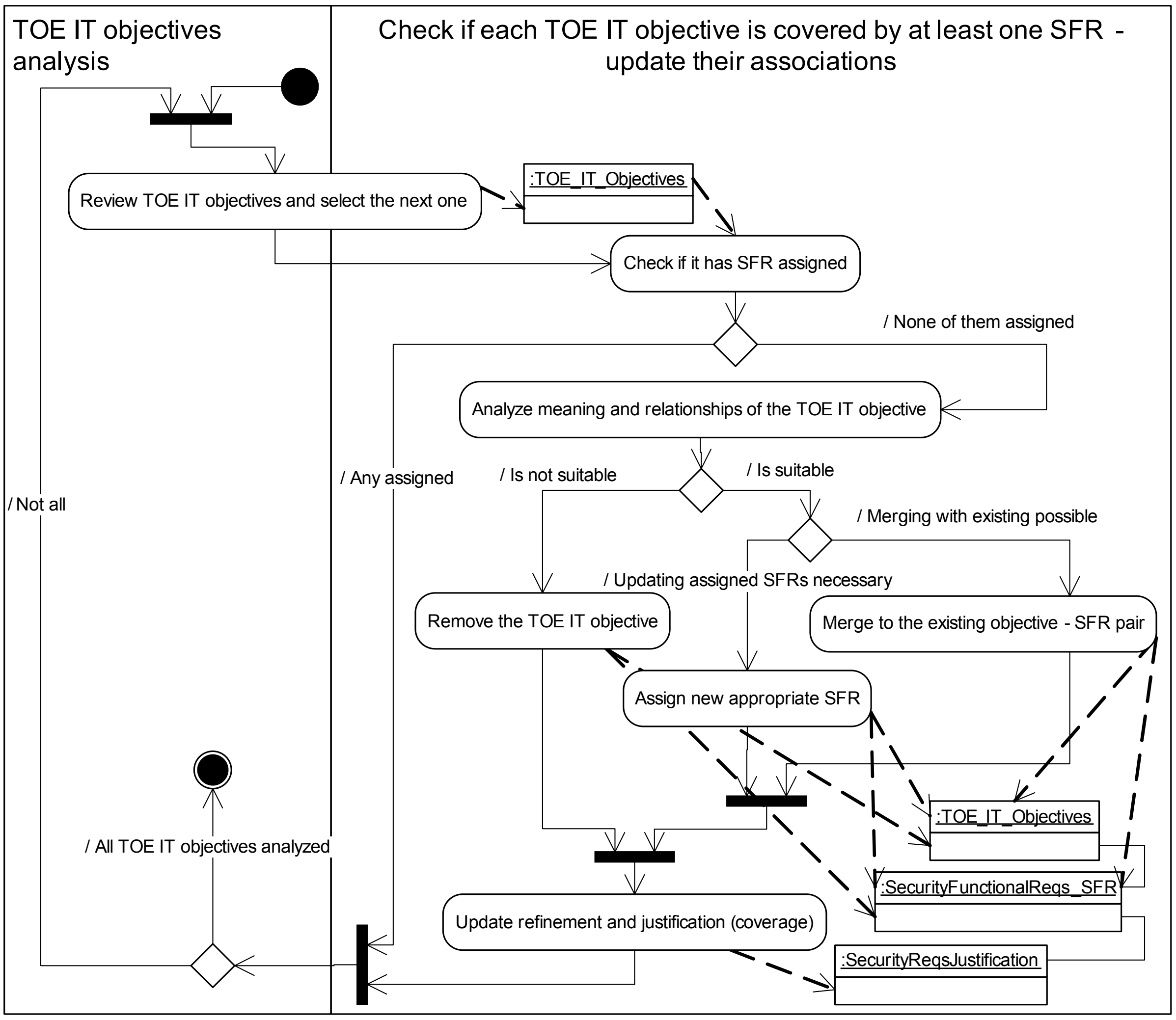

TOE IT objectives analysis
Check if each TOE IT objective is covered by at least one SFR - update their associations
Review TOE IT objectives and select the next one
:TOE_IT_Objectives
Check if it has SFR assigned
/ None of them assigned
Analyze meaning and relationships of the TOE IT objective
/ Any assigned
/ Is not suitable
/ Is suitable
/ Not all
/ Merging with existing possible
/ Updating assigned SFRs necessary
Remove the TOE IT objective
Merge to the existing objective - SFR pair
Assign new appropriate SFR
:TOE_IT_Objectives
/ All TOE IT objectives analyzed
:SecurityFunctionalReqs_SFR
Update refinement and justification (coverage)
:SecurityReqsJustification

Fig. 10.9. The security requirements rationale – revealing uncovered security objectives
Rys. 10.9. Uzasadnienie wymagań bezpieczeństwa przedmiotu oceny – identyfikacja celów na zabezpieczenia nie pokrywanych przez wymagania

Let us go back to the main activity diagram shown in the Fig. 10.6. After completing the functional security requirements rationale, i.e. SFRs checkings, the next main step of the whole rationale process is SARs checking, presented in the Fig. 10.10.

The SARs must meet security objectives too, must not be excessive (to minimize the costs and time scale of solutions) and must be possible to satisfy during implementation.

The Fig. 10.10 also presents SOF claims checking. They must be appropriate, mainly with respect to the presumed attackers' potential. Please note that the SOF claims help to qualify security functions built-in to the TOE.

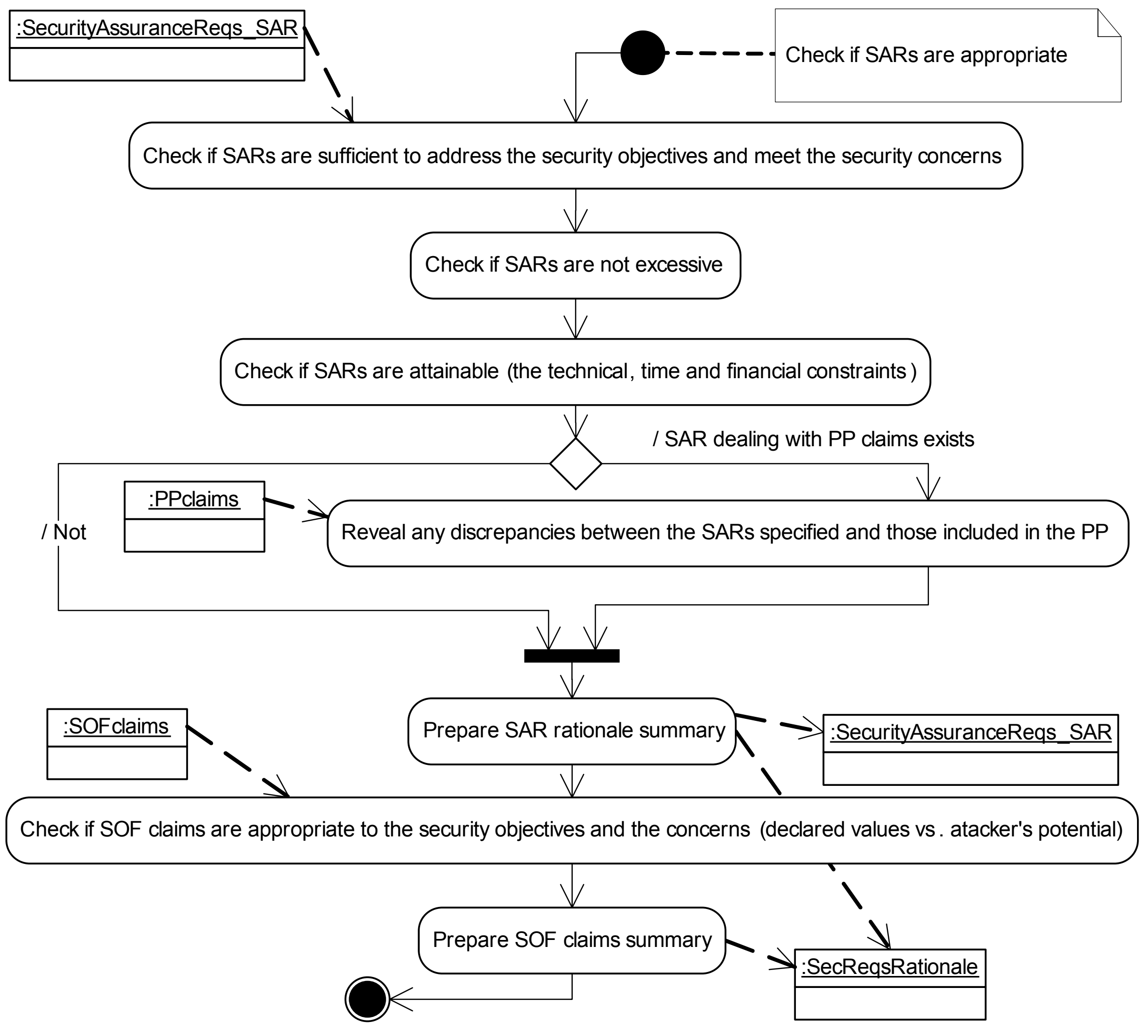

:SecurityAssuranceReqs_SAR
Check if SARs are appropriate
Check if SARs are sufficient to address the security objectives and meet the security concerns
Check if SARs are not excessive
Check if SARs are attainable (the technical, time and financial constraints)
/ SAR dealing with PP claims exists
:PPclaims
/ Not
Reveal any discrepancies between the SARs specified and those included in the PP
:SOFclaims
Prepare SAR rationale summary
:SecurityAssuranceReqs_SAR
Check if SOF claims are appropriate to the security objectives and the concerns (declared values vs. atacker's potential)
Prepare SOF claims summary
:SecReqsRationale

Fig. 10.10. The security requirements rationale – SARs and SOF claims checking
Rys. 10.10. Uzasadnienie wymagań bezpieczeństwa przedmiotu oceny – sprawdzenie wymagań uzasadniających zaufanie (SAR) oraz deklaracji siły funkcji zabezpieczających (SOF)

These claims express the minimum efforts assumed necessary to defeat the TOE expected security behaviour when a direct attack to its underlying security mechanisms is performed by an intruder. Three predefined SOF levels, as “security functions resistance measures”, are placed in the Appendix A.

The last step is to demonstrate that the specified set of requirements creates a kind of an integrated whole. Mutual supportiveness demonstration is shown in the Fig. 10.11.

Mutual supportiveness guarantees consistency of the applied countermeasures.

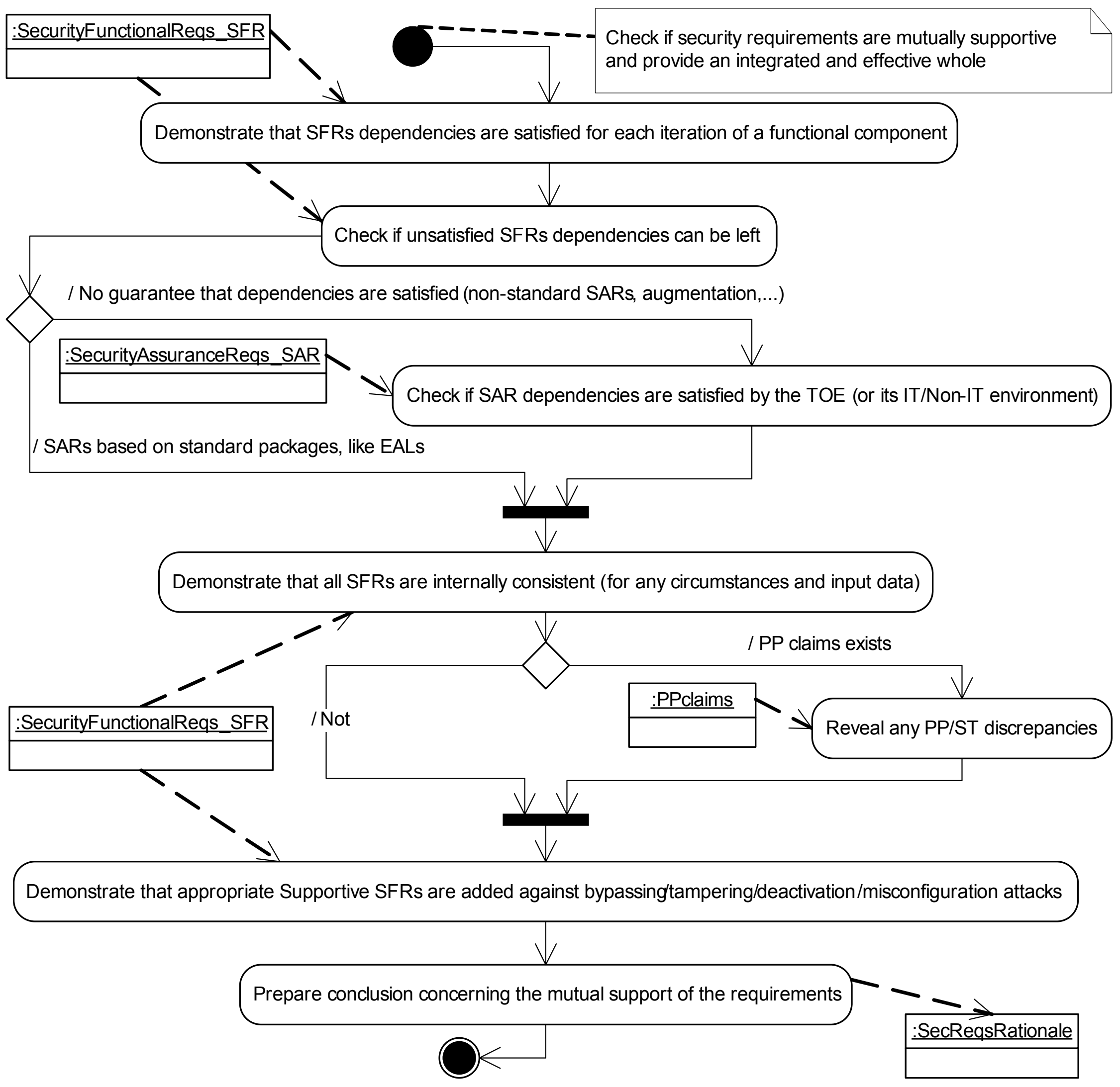

:SecurityFunctionalReqs_SFR
Check if security requirements are mutually supportive and provide an integrated and effective whole
Demonstrate that SFRs dependencies are satisfied for each iteration of a functional component
Check if unsatisfied SFRs dependencies can be left
/ No guarantee that dependencies are satisfied (non-standard SARs, augmentation,...)
:SecurityAssuranceReqs_SAR
Check if SAR dependencies are satisfied by the TOE (or its IT/Non-IT environment)
/ SARs based on standard packages, like EALs
Demonstrate that all SFRs are internally consistent (for any circumstances and input data)
/ PP claims exists
:SecurityFunctionalReqs_SFR
/ Not
:PPclaims
Reveal any PP/ST discrepancies
Demonstrate that appropriate Supportive SFRs are added against bypassing/tampering/deactivation/misconfiguration attacks
Prepare conclusion concerning the mutual support of the requirements
:SecReqsRationale

Fig. 10.11. The security requirements rationale – checking if the security requirements are mutually supportive
Rys. 10.11. Uzasadnienie wymagań bezpieczeństwa przedmiotu oceny – wykazanie, że wymagania bezpieczeństwa wspomagają się wzajemnie

Please note that due to the coherency (by definition) of defined assurance packages, SARs from such packages are easier to use.

10.3. TOE summary specification rationale

The elaboration of the TOE summary specification rationale, i.e. the definition of the behaviour of the `TSSRationale` class (Fig. 10.1) is presented in the Fig. 10.12. This class works on the same data structures as the `TSSElaboration` class does, providing the optimization their contents only. The activity diagram for the TOE summary specification rationale, shown in the Fig. 10.12, refines the following steps of the TSS rationale:

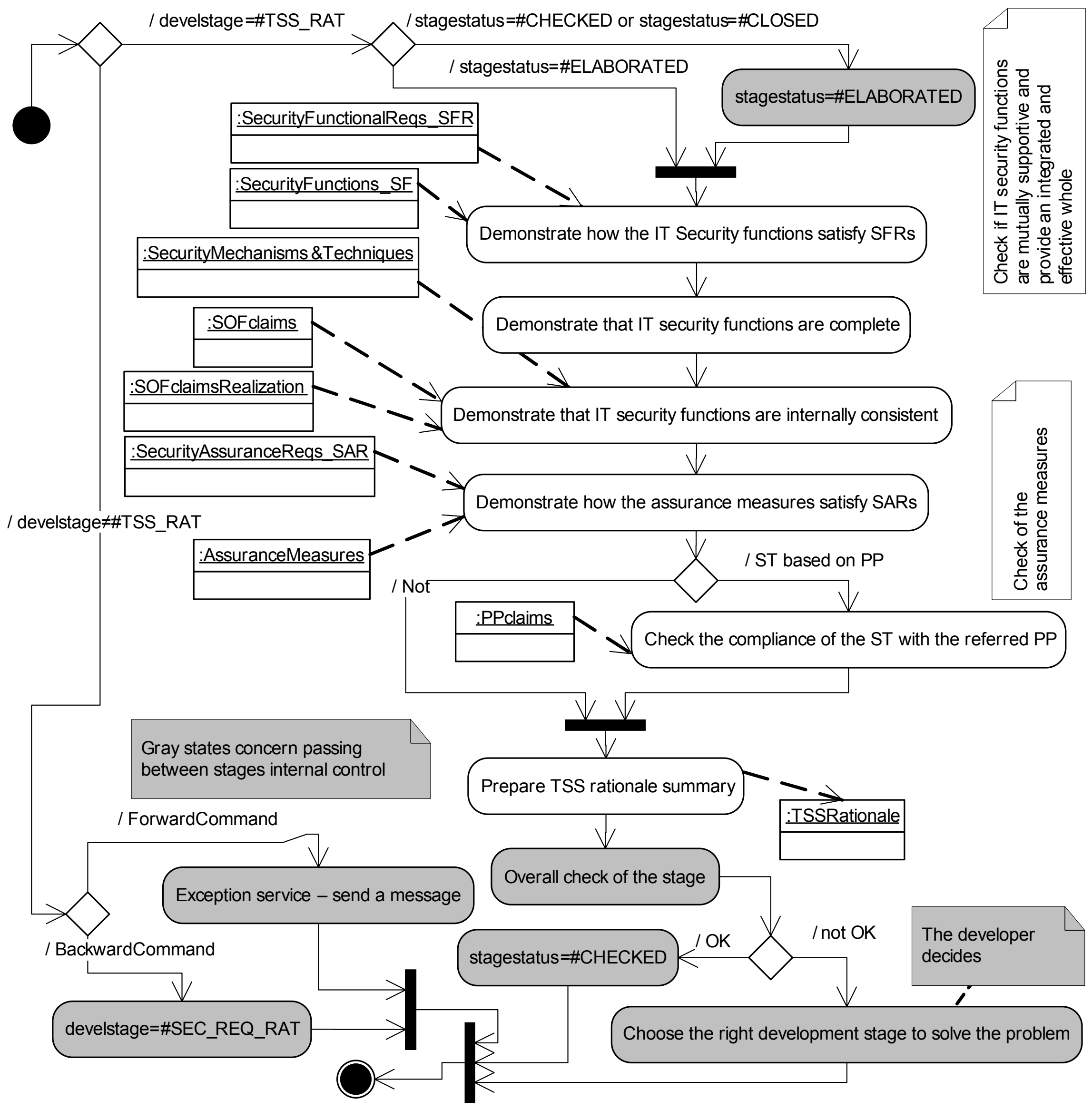

/ develstage=#TSS_RAT
/ stagestatus=#CHECKED or stagestatus=#CLOSED
/ stagestatus=#ELABORATED
stagestatus=#ELABORATED
Check if IT security functions are mutually supportive and provide an integrated and effective whole
:SecurityFunctionalReqs_SFR
:SecurityFunctions_SF
Demonstrate how the IT Security functions satisfy SFRs
:SecurityMechanisms &Techniques
Demonstrate that IT security functions are complete
:SOFclaims
:SOFclaimsRealization
Demonstrate that IT security functions are internally consistent
:SecurityAssuranceReqs_SAR
Demonstrate how the assurance measures satisfy SARs
Check of the assurance measures
/ develstage≠#TSS_RAT
:AssuranceMeasures
/ ST based on PP
/ Not
:PPclaims
Check the compliance of the ST with the referred PP
Gray states concern passing between stages internal control
Prepare TSS rationale summary
:TSSRationale
/ ForwardCommand
Exception service – send a message
Overall check of the stage
/ BackwardCommand
stagestatus=#CHECKED
/ OK
/ not OK
The developer decides
develstage=#SEC_REQ_RAT
Choose the right development stage to solve the problem

Fig. 10.12. The TSS rationale – the general activity diagram
Rys. 10.12. Uzasadnienie końcowej specyfikacji przedmiotu oceny (TSS) – nadrzędny diagram czynności

- demonstration: how IT security functions satisfy the SFRs (should be necessary and sufficient, explanation how particular SFRs are satisfied, with respect to SOF claims, security mechanisms and techniques);
- consideration: how assurance measures satisfy SARs;
- for ST based on PP, checking compliance with the referred PP.

Please note that the demonstration how IT security functions satisfy SFRs is performed in a similar way as it was shown in the Fig. 10.8 and 10.9.

10.4. Formal approach to the security target rationale

The rationale process concerns the entire security target (ST) and is based on the TOE model elaborated during all IT security development stages. Let us introduce the security target model definition, which extends earlier definitions: 5.1, 6.1, 7.1, 8.1.

Definition 10.1: Security target.

The security target specification (ST) is the triplet of sets:

$S_{ST} = <S_{ST\_Items}, S_{ST\_Pars}, S_{ST\_Assocs}>$.

1. The first element of the triplet, S_{ST_Items} is the sum of three disjointed sets of generics, functional or assurance components:

$S_{ST\_Items} = S_{TOESecEnvItems} \cup S_{SecObjItems} \cup S_{SecReqsItems} \cup S_{TSS\_SecFuns}$ and

$S_{TOESecEnvItems} \cap S_{SecObjItems} \cap S_{SecReqsItems} \cap S_{TSS\_SecFuns} = \varnothing$.

2. The second element of the triplet, S_{ST_Pars} is the sum of three sets of parameterization associations (encompassing all of them):

$S_{ST\_Pars} \subseteq \mathsf{GenParAssoc}$ and $S_{ST\_Pars} = S_{TOESecEnvPars} \cup S_{SecObjPars} \cup S_{SecReqsPars}$.

3. The final element of the triplet, S_{ST_Assocs} is the sum of three sets of associations concerning mapping (encompassing all of them):

$S_{ST\_Assocs} \subseteq \mathsf{SecAssoc}$ and $S_{ST\_Assocs} = S_{SecObjAssocs} \cup S_{SecReqsAssocs} \cup S_{TSS\_SecAssocs}$.

□

Definition 10.2: Security target elaboration.

The security target elaboration is the stepwise transformation of $S_{TOESecEnv}$ to S_{SecObj}, S_{SecObj} to $S_{SecReqs}$, and finally $S_{SecReqs}$ to S_{TSS}, i.e. transformation of the corresponding items included in S_{ST_Items}, with their parameters expressed by S_{ST_Pars}, and according to the mapping rules represented by the S_{ST_Assocs} associations.

□

This definition extends earlier definitions: 6.2, 7.2 and 8.2. Now the modelled specifications are refined, optimized and checked. In this case checkings are especially important if one group of items covers properly other group of items and redundant items do not exist. The introduced navigation facilities through all stages can also be helpful while the coverage analyses are performed. These facilities were implemented into the computer-aided tool that will be shown later.

The UML models of the TOE security target are rather complex structures, even if only several threats/OSPs are identified. For this reason, the simplified models can be presented with their key aspects only, which is allowed in the UML approach.

The Fig. 10.13 shows one of such examples. In the upper part of the figure there is the main `ST_Elaboration` class with its subclasses responsible for particular stages. Below, there are classes representing specification elements for four stages encompassed by the security target rationale.

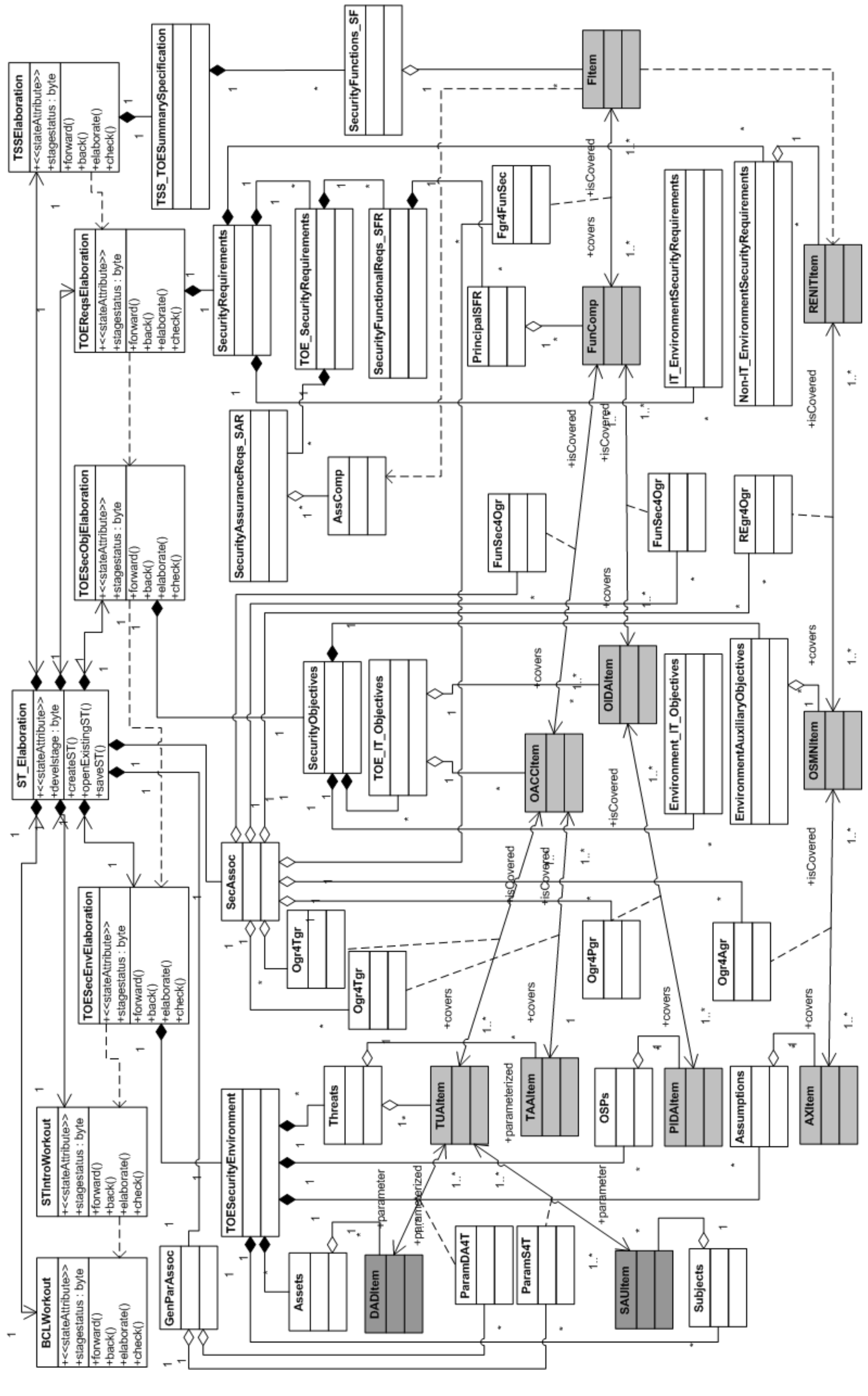


Fig. 10.13. The security target rationale – a summary example

Rys. 10.13. Uzasadnienie zadania zabezpieczeń – przykład podsumowujący

Three supporting chains consisting of the elements (light grey) of the specification are presented on a very general level (i.e. the family level). The first chain starts from two threats, one of the TAA family item, the other of the TUA family item. These threats are covered by the common `OACCItem` objective item, for which the `FunComp` component is proposed and implemented by the `FItem` security function. The second begins at the `PIDAItem` policy rule, satisfied by the `OIDAItem` objective, which is covered, together with the mentioned `OIDAItem`, by the `FunComp`. The third chain starts on the AX assumption, satisfied by the organizational means represented by the `OSMNItem`, which leads to the `RENITItem` requirement.

Please note some examples of the associations responsible for mapping the security issues, forming these chains. All association classes are grouped by the main SecAssoc class.

Additionally, one, rather symbolic, assurance requirement is added. Let it be one of the EAL2 package components, for example.

Only one item, i.e. the `TUAItem`, is parameterized while others are left intentionally uncompleted. The parameterization associations are grouped by the GenParAssoc class. Corresponding `DADItem` and `SAUItem` are marked with dark grey class symbols on the left side.

This model is simplified because it operates on any items of the given family. Each of these items should be a concrete and unique item or component (Fig. 3.21, Fig. 3.22). Parameterization and mapping are shown on a very general level too. Each connection of chain elements is carried out by a concrete association whose properties are represented by those shown in the discussed figure. The role names, such as isCovered, covers, parameter, parameterized, represent the sets of unique role names derived from the unique generic or component item names.

Even on this general level different model properties can be identified or checked, and different constraints can be set using the OCL approach.

Example 10.1: Identifying the threats that are opposed by the given security function, represented by the `FItem`.

This is backwards navigation (horizontal) started on `FItem` through a supporting chain (i.e.: `TUAItem`, `TAAItem` – `OACCItem` – `FunComp` – `FItem`). It is implemented by the visualisation facility that will be shown later, i.e. when computer-aided tool is presented.

```
FItem
self.funComp.oACCItem.associationEnds
   ->select(c:TgrGeneric)
```

This expression can be used to define an operation returning the set of all threats. In this case (Fig. 10.13) only some real `TUAItem` or `TAAItem` items will be identified. □

Example 10.2: Identifying how and where the given OSP rule is implemented.

This is forward navigation (horizontal) through a supporting chain started from the given OSP rule represented by `PIDAItem` and ended on `FItem`. The chain can also be visualized by the mentioned visualization facility.

```
PIDAItem
self.oIDAtem.funComp.associationEnds
   ->select(c:FgrGeneric)
```

This expression can be used to define an operation returning the set of all security functions enforcing this policy rule. In this case it can be any object represented by the `FItem` class. □

Example 10.3: Identifying items of the considered type used by the developer who prepares the security target specification. Let us identify all REPH-family generics that can be used only for the IT environment requirements specification.

This is navigation (vertical) through the given stage of the model starting from the "root" , i.e. the `ST_Elaboration` class.

```
ST_Elaboration
self.tOEReqsElaboration.securityRequirements.
   iT_EnvironmentSecurityRequirements
   ->select(c:REPHGen)
```

Please note that adding "`->size`" instead of the "`->select`" at end of the expression allows to count these items.

□

More complicated operations can be carried out with the use of more advanced operations on collections, e.g. iteration or intersection. Some of these possibilities were presented earlier, during the discussion on particular development stages.

In some circumstances this level of abstraction seems to be superficial, especially for discussing details of the computer-aided tool, which will be presented later. The Fig. 10.13 operates on the generic and or component families representatives, i.e. "items". The OCL expressions allow to look at the model on a more detailed level. This will be exemplified with the reference to this figure which shows a part of the real model on a very general level. Let us proceed to a more detailed level to identify the real specification items and relations between them. We can obtain results as those shown in the following examples:

Example 10.4: Identifying specification items of the considered family used by the ST developer.

Different operations can be defined in the context of the given family, e.g. for TAA family:

```
TAAItem
self->select(c:TAAGen).associationEnds
-- returns set of TAAItem generics represented
-- by the TAAItem, e.g. the set:
```

`Set(TAAItem)`=Set{*ErrOmit_D0I0, ModifyData_D0I0, NeglComprAss_D1I0*},
representing the following real generic items:

TAA.ErrOmit_D0I0. Administrative errors of omission.

TAA.ModifyData_D0I0. Hostile administrator's modification of user or system data.

TAA.NeglComprAss_D1I0. Compromise of IT assets may occur as a result of actions taken by careless, wilfully negligent or hostile administrators or other privileged users [paramSAU].

This operation on other families returns respectively:
`Set(TUAItem)`=Set{*DataRecDenied_D0I0, DataOwnsDenied_D0I0, ConsData_D0I0, ErrSecurity_D0I0, KeyCorrupted_D0I0, VirusComprAss_D0I0*}, representing the following generics:

TUA.DataRecDenied_D0I0. Recipient [paramSAU] denies receiving information [paramDAD].

TUA.DataOwnsDenied_D0I0. Data ownership of [paramDAD] is denied by an authorized [paramSAU].

TUA.ConsData_D0I0. An authorized user [paramSAU] of the TOE consumes global resources [paramDAS], in a way which compromises the ability of other authorized users to access or use those resources.

TUA.ErrSecurity_D0I0. User errors undermine the system's security features.

TUA.KeyCorrupted_D0I0. Private key was compromised, i.e. it was dropped by another authorized system user.

TUA.VirusComprAss_D0I0. Compromise of the integrity and/or availability of IT assets may occur as a result of an authorized user [paramSAU] of the TOE unwittingly introducing a virus into the system.

`Set(OACCItem)`=Set{*MaintenanceAcc_D0I0, MaintenanceRec_D0I0, AdmLmtAcc_D0I0, DAC_D0I0*}, expressing the items:

OACC.MaintenanceAcc_D0I0. Controlling access to the system by maintenance personnel who troubleshoot the system and perform system updates.

OACC.MaintenanceRec_D0I0. Automatic termination of the privilege of user access to system maintenance, after the expiration of assigned timed interval.

OACC.AdmLmtAcc_D0I0. Limitation of administrative access control.

OACC.DAC_D0I0. The TOE will provide its users with the means of controlling and limiting access to the objects and resources they own or are responsible for, on the basis of individual users or identified groups of users, and in accordance with the set of rules defined by the security policy.

`Set(OIDAItem)`=Set{*UserAuthManage_D3I3*, *AdmLmtBind_D0I6*, *AdmAttMod _D0I0*, *RestrUserEntry_D0I0*, *AuthDataSafe_D0I5*}, representing the items:

OIDA.UserAuthManage_D3I3. User authorization management.

OIDA.AdmLmtBind_D0I6. Limiting an administrator's ability to modify user-subject bindings.

OIDA.AdmAttMod_D0I0. Limiting an administrator's modification of user attributes.

OIDA.RestrUserEntry_D0I0. The TOE will have the capability of restricting user entry to itself based on the time and entry of the device location.

OIDA.AuthDataSafe_D0I5. Those responsible for the TOE must ensure that the authentication data for each user account for the TOE is held securely and not disclosed to persons not authorized to use that account.

`Set(FunComp)`=Set{*FAU_ARP.1_I2*, *FAU_GEN.1_I0*, *FAU_SAA.1_I3*, *FMT_MSA.1_I0, FIA_ATD.1_I0*}, representing the examples of the functional components:

FAU_ARP.1_I2 Security alarms.
FAU_GEN.1_I0 Audit data generation.
FAU_SAA.1_I3 Potential violation analysis.
FMT_MSA.1_I0 Management of security attributes
FIA_ATD.1_I0 User attribute definition.

`Set(FItem)=` Set{*CtrlConf_D7I0*, *DataConfid_D3I0*}, representing the examples of the real security functions:

CtrlConf_D7I0. Control&Configuration module will provide work configuration, certificates validity checking and CSP selection.

DataConfid_D3I0. Data confidentiality module will provide confidentiality for protected information by using cryptographic functions. □

In the real designs there exist not only numerous generic or component items but also complicated relationships between them. Please note that both kinds of the associations (GenParAssoc, SecAssoc) are defined on the group level, catching common features of both. In the real designs real associations between the specification items exist. These associations inherit properties of GenParAssoc, SecAssoc, defined on a general level. Let us focus on the SecAssoc examples.

For any of the SecAssoc group where SecAssoc = ⟨*Ogr4Tgr*, *Ogr4Pgr*, *Ogr4Agr*, *FunSec4Ogr*, *AssSec4Ogr*, *REgr4Ogr*, *Fgr4FunSec*⟩ there exist many of the associates functions as (Definition 3.51). They express association properties on a more detailed level which has not been discussed yet. The associates functions will be shown using only few examples. The full list of these relationships is built in the design library (`SL_SecurityLibrary` class) and they function there as “proposed”, but may change their status to “mapped”. Please note that the function associates maps each association name $as \in$ ASSOC to a finite list of classes $c_1, \ldots, c_n$ participating in the association:

$as \mapsto \langle c_1, \ldots, c_n \rangle$, with $(n \geq 2)$.

Using the association names convention, we can define associations on the family level, e.g.: $as^{i}_{OACC_4TAA}$, $as^{i}_{OIDA_4TAA} \in$ *Ogr4Tgr* and others, mentioned below. Each of them is numbered. The number i can be interpreted as the instance number.

Example 10.5: Security associations concerning mapping.

Identifying specification items of the considered family used by the ST developer.
Let us consider the following relationships between the TAA- and OACC- families. Please note the intentional inconsistency concerning names – the Common Criteria developers convention is used to distinguish generic families.

$as^{1}_{OACC_4TAA} \mapsto$
⟨ *TAA.ErrOmit_D0I0, OACC.MaintenanceAcc_D0I0, OACC.MaintenanceRec_D0I0* ⟩,

$as^{2}_{OACC_4TAA} \mapsto$
⟨ *TAA.ModifyData_D0I0, OACC.AdmLmtAcc_D0I0* ⟩,

$\mathsf{as}^3{}_{\mathsf{OACC4TAA}} \mapsto$ ⟨ *TAA.NeglComprAss_D0I0, OACC.DAC_D0I0* ⟩ ,
and between TAA- and OIDA- families:
$\mathsf{as}^1{}_{\mathsf{OIDA4TAA}} \mapsto$ ⟨ *TAA.ErrOmit_D0I0, OIDA.UserAuthManage_D0I0* ⟩ ,
$\mathsf{as}^2{}_{\mathsf{OIDA4TAA}} \mapsto$
⟨ *TAA.ModifyData_D0I0, OIDA.AdmLmtBind_D0I0, OIDA.AdmAttMod_D0I0* ⟩ ,
$\mathsf{as}^3{}_{\mathsf{OIDA4TAA}} \mapsto$
⟨ *TAA.NeglComprAss_D0I0, OIDA.RestrUserEntry_D0I0, OIDA.AuthDataSafe_D0I0* ⟩ .

For the functional components relationships the as functions may concern both component classes or families.
$\mathsf{as}^1{}_{\mathsf{FMT\_MSA4OACC}} \mapsto$
⟨ *OACC.MaintenanceAcc_D0I0, OACC.MaintenanceRec_D0I0, FMT_MSA.1_I0* ⟩ ,
$\mathsf{as}^1{}_{\mathsf{FAU4OACC}} \mapsto$
⟨ *OACC.AdmLmtAcc_D0I0, FAU_ARP.1_I2, FAU_GEN.1_I0, FAU_SAA.1_I3* ⟩ ,
$\mathsf{as}^2{}_{\mathsf{FAU4OACC}} \mapsto$
⟨ *OACC.DAC_D0I0, FAU_ARP.1_I2, FAU_GEN.1_I0, FAU_SAA.1_I3* ⟩ ,
$\mathsf{as}^1{}_{\mathsf{FIA4OACC}} \mapsto$
⟨ *OACC.DAC_D0I0, FIA_ATD.1_I0* ⟩ ,

Additionally, detailed mapping associations may be defined between the functional requirements and security functions (usually between the grouped principal requirements and functions implementing them):
$\mathsf{as}^1{}_{\mathsf{F4SFR}} \mapsto$
⟨ *FAU_ARP.1_I2, FAU_GEN.1_I0, FAU_SAA.1_I3, F.CtrlConf_D7I0* ⟩ ,
$\mathsf{as}^2{}_{\mathsf{F4SFR}} \mapsto$
⟨ *FMT_MSA.1_I0, FIA_ATD.1_I0, F.DataConfid_D3I0* ⟩ .
Two associations were defined there, each for one function (components selection is rather hypothetical).
□

For these associations their role names (Definition 3.52) can be identified in a simple way, using generic or component names. All multiplicities must have value {1,1,....,1} due to the uniqueness of the item names.

Example 10.6: Role names.

Let us consider the first and the last association from the above example:
$\mathsf{as}^1{}_{\mathsf{OACC4TAA}} \mapsto$
⟨ *TAA.ErrOmit_D0I0, OACC.MaintenanceAcc_D0I0, OACC.MaintenanceRec_D0I0* ⟩ ,
$\mathsf{as}^2{}_{\mathsf{F4SFR}} \mapsto$
⟨ *FMT_MSA.1_I0, FIA_ATD.1_I0, F.DataConfid_D3I0* ⟩ .
The role names can be defined using unique class names:
$\mathsf{roles}(\mathsf{as}^1{}_{\mathsf{OACC4TAA}}) =$
⟨ *errOmit_D0I0, maintenanceAcc_D0I0, maintenanceRec_D0I0* ⟩ ,

roles(as^2_{F4SFR}) =
⟨*fMT_MSA.1_I0, fIA_ATD.1_I0, dataConfid_D3I0*⟩ .

The role names can be assigned to the isCovered/covers sets on the group level:

{*errOmit_D0I0*} ∈ Set(isCovered), while
{*maintenanceAcc_D0I0, maintenanceRec_D0I0*} ∈ Set(covers).
{*fMT_MSA.1_I0, fIA_ATD.1_I0*} ∈ Set(isCovered), while
{*dataConfid_D3I0*} ∈ Set(covers).

□

For the given specification item, all associations in which the item participates can be found using the participating function (Definition 3.54).

Example 10.7: The coverage analyses with the use of the participating function.

For the given specification item, e.g. *OACC.DAC_D0I0*, all items covered by it and all items that it covers can be found using this function:

participating(*OACC.DAC_D0I0*) = {$as^3_{OACC4TAA}$, $as^2_{FAU4\,OACC}$, $as^1_{FIA4OACC}$}.

This allows to navigate starting from *TAA.NeglComprAss_D0I0* ∈ $as^3_{OACC4TAA}$, through *OACC.DAC_D0I0*, to *FIA_ATD.1_I0* ∈ $as^1_{FIA4OACC}$. Finding association concerning the *FIA_ATD.1_I0* component allows to reach the security function used to the opposite *TAA.NeglComprAss_D0I0*:

participating(*FIA_ATD.1_I0*) = {$as^1_{FIA4OACC}$, as^2_{F4SFR}}.

Using more sophisticated functions, i.e. navends(*OACC.DAC_D0I0*), this result can be obtained in one step. However, other associations (outside the considered supporting chain) will be returned as the result too.

□

11. IT SECURITY SELF-EVALUATION FRAMEWORK

IT security products or systems, developed with the use of the framework presented there, should pass to the evaluation process performed by independent bodies. Evaluators working in these labs use their own evaluation frameworks based on the evaluation scheme and related methodologies, for example presented in [45-46], [41].

The presented IT Security Development Framework, although intended to be used by designers, has a simple self-evaluation framework built in, compliant with the above mentioned standards. The developer has a possibility:

- to see how his/her product or system will be evaluated,
- to prepare a proper set of evidence material, to make corrections as soon as possible.

This self-evaluation framework may be adopted to security evaluation labs needs. The IT security self-evaluation framework expressed by the SEM model is, together with the product model (PM) and security model (SM), the third part of the presented Common Criteria compliant IT Security Development Framework (Fig. 2.3). The SEM model is shown in the Fig. 11.1 as a class diagram. The self-evaluation model must comply with the chosen evaluation methodology (CEM) [45-46] and the evaluation scheme [41]. The CEM recognizes three mutually exclusive verdict types:

- `PASS` verdict means the completion of the CC evaluator action element and determination that the requirements for the PP, ST or TOE under evaluation are satisfied;
- `FAIL` verdict means that the CC evaluator action element is performed, but the requirements for the PP, ST, or TOE under evaluation are not met;
- `INCONCLUSIVE` verdict is defined to indicate that evaluation is under work; all verdicts have this value assigned at the beginning of the evaluation and remain so until either a `PASS` or `FAIL` verdict is determined.

During the evaluation process three types of formalized reports may be created. The first one, the Observation Report (OR), expresses partial results of work performed during the evaluation. It contains a problem, an observation or decisions submitted to the developers or received from them by the security lab. In this case, during evaluation performed by the developer, the report can be used to document the problems and their solutions.

The Evaluation Discovery Report (EDR) contains more detailed information on groups of security issues and results of work performed during the evaluation (a work package [46] in which the problem was discovered, a brief summary of the problem, and their status).

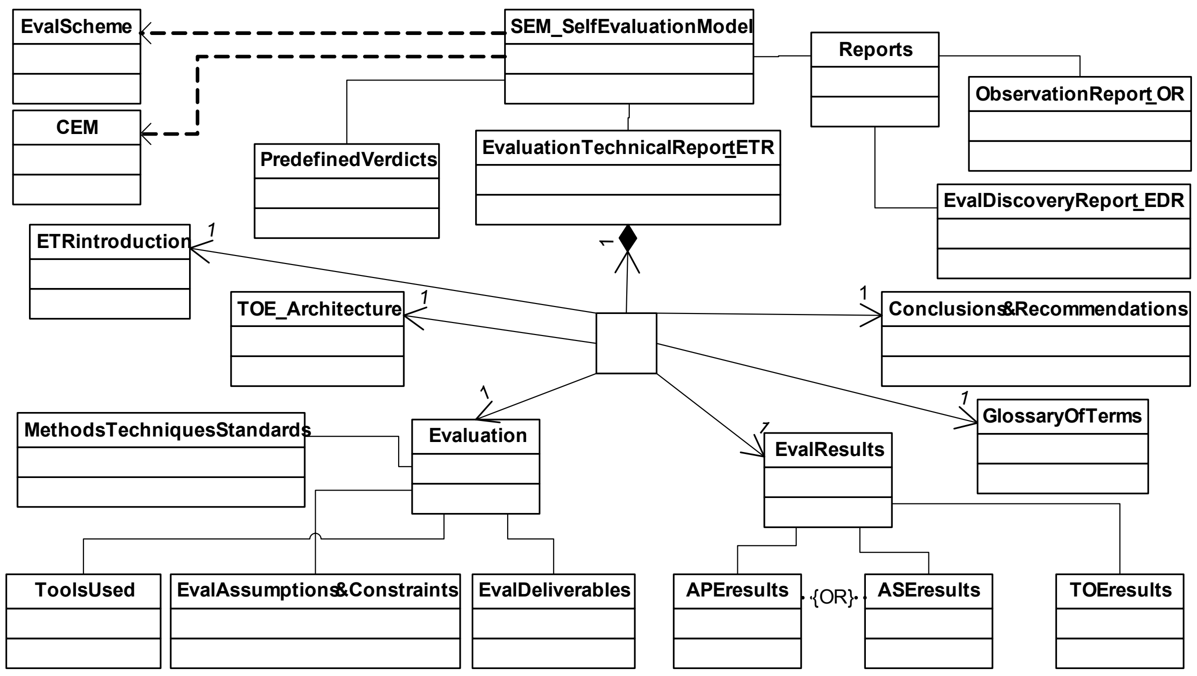


Fig. 11.1. IT security self-evaluation framework model (SEM)
Rys. 11.1. Model podsystemu szkieletowego (SEM) do prowadzenia samodzielnej oceny zabezpieczeń

The Evaluation Technical Report (ETR) is the main document summarizing the evaluation process. It begins with an introduction containing all identifiers which unambiguously identify the TOE and all references. Architectural description of the TOE class provides a high level description of the IT product or system, its main components and degree of architectural separation. This ETR section is based on the deliverables specified according to the Common Criteria assurance family called Development High Level Design (ADV_HLD).

The `Evaluation` class specifies the evaluator's workshop, containing:

- methods, techniques and standards dealing with the evaluation criteria, methodology and interpretations used to evaluate the TOE or devices used to perform the tests,
- evaluation tools comprising the software and hardware tools supporting the evaluation process,
- assumptions and constraints, like: constraints on the evaluation, constraints on the distribution of evaluation results and assumptions made during the evaluation that have an impact on the evaluation results, information in relation to legal or statutory aspects, organization, confidentiality, etc.,

- evaluation deliverables, encompassing a set of evaluation evidence and participating bodies (e.g. the developer, the sponsor) and other identifiers.

The main part of the ETR is Evaluation results section (`EvalResults` class), which contains the verdicts and their rationales for each of the evaluated assurance components. The rationale justifies the verdict using the CC, CEM, any interpretations and the evaluation evidence analyzed. It shows how the evaluation evidence does or does not meet each aspect of the criteria and contains a description of the work performed, the method used, and any derivation of results.

As it will be shown in the chapter 12 and in the Appendix E (Example E.6), every assurance component has three types of elements, dealing with:

- developer's delivered evidence material (D),
- contents and presentation of the evidence material (C),
- evaluator's action performed to check if the evidence material is delivered, and if it has right content and presentation (E).

Please note that the most granular CC structure to which a verdict (`PASS`, `FAIL`, `INCONCLUSIVE`) can be assigned is the evaluator action element (E). The overall evaluation verdict is `PASS` only if all the constituent verdicts are also `PASS`.

The framework encompasses all possible kinds of self-evaluation:

- ST self-evaluation according to the ASE (Security Target evaluation assurance class), focused on the ST specification checking, or
- PP self-evaluation according to the APE (Protection Profile evaluation assurance class), focused on the PP specification checking,
- TOE self-evaluation for the required EAL with its augmentation or substitution, deals with product or system security features.

First, the ST or PP self-evaluation should be done, then the TOE self-evaluation is performed. Conclusions and recommendations provide evaluation results summary, remarks and suggestions that may be useful for the overseer, including the detected shortcomings of the IT product or system or a mention of features which are particularly useful. The glossary of terms section contains all acronyms, abbreviations and terms used in the ETR.

The main use of the built-in self-evaluation framework is to perform self-evaluation of the developed product or system, particularly to check how the prepared evidence material satisfies assurance components. As this work deals with security development, the TOE evaluation process was presented in a very concise way. The part of the tool that supports evaluators can work separately from the developers' part. Designers who are more familiar with the evaluation process have better understanding how their works will be independently verified, and will be able to express their concepts within ST or PP documents more adequately. The presented tool allows to support the whole evaluation process for Security Targets, Protection Profiles, and the TOE vs. the claimed EAL.

12. IMPLEMENTATION AND EVALUATION OF THE FRAMEWORK

The research phase of the UML framework for IT security development encompassed: studying best practices, standards, publications, research papers, and discussing the topic with experts [13], [19], [28]. Then some case studies were performed, like: analyzing the existing PP and ST, experiences with the computer-aided IT security development and evaluation tools that were developed before or concurrently, and carrying out our own projects of the discussed framework and tool.

There are two aspects of using the developed ITSDF framework. First, it can be used by engineers manually (with the help of standard text editors) – all data models are design patterns, and activity diagrams can be considered as design procedures. It forms some kind of guidelines for CC developers. Their UML representation facilitates and makes more coherent the whole IT security development process. Additional advantages can be achieved going one step farther, and creating a computer-aided tool on this basis. It will make these activities easier by managing the entire IT security development or evaluation processes, by models reusability, by graphical support, providing statistical data, reporting wizards and documentation management, etc. This real technology transfer phase deals not only with the development of the ITSDF-tool, but also with presenting it to specialists and performing validation in the IT security development and evaluation labs.

During the tool development the following assumptions are taken into consideration:

- Common Criteria and related standards compliance,
- development and evaluation processes support,
- flexibility (openness to the standards modification, new technologies and methodologies).

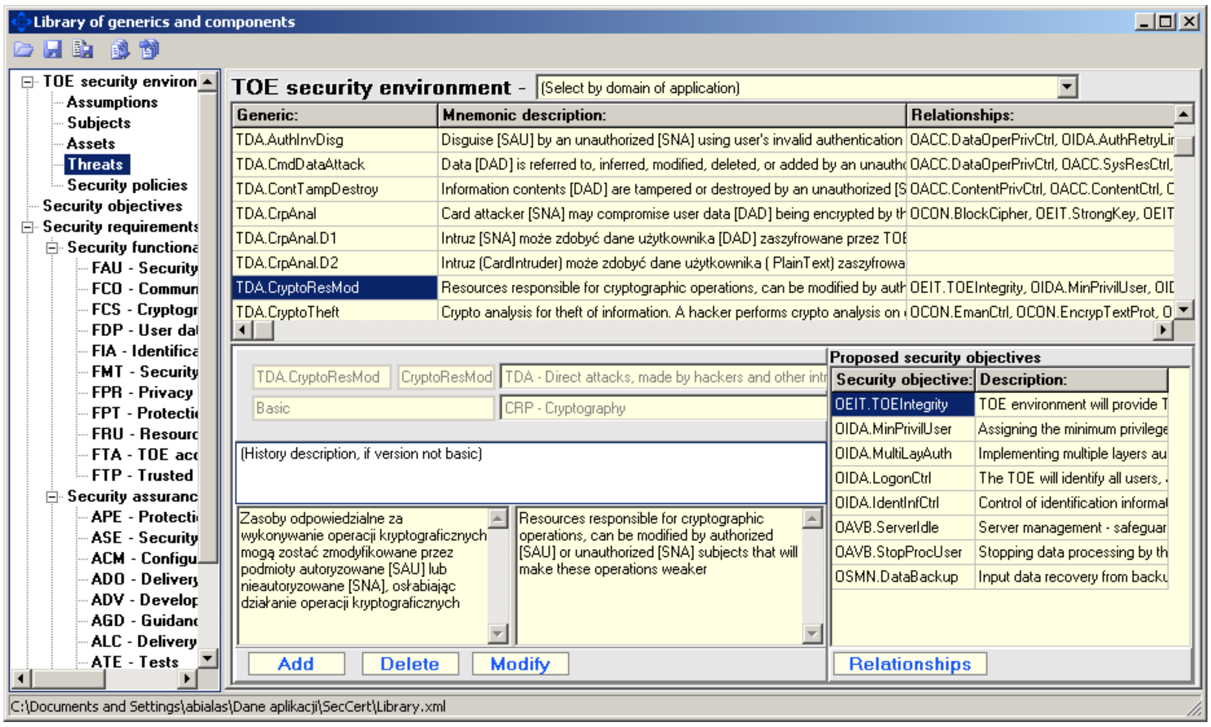
The UML framework for IT security development was validated mostly on the COTS-type digital signature/encryption application, based on the Microsoft CryptoAPI®, called SecOffice [95], [90] and on the existing security targets (like Philips smart card controller [87-89]), and also on protection profiles examples. Please refer to the firewall example presented in the Appendix E. Verification of compliance with [60] was provided too.

These works will be exemplified below using a few representative ITSDF-tool screen shots.

12.1. Generics and components library

The open set of parameterized generics was defined and implemented as a ITSDF-tool library, allowing to specify different aspects of IT security for a large group of TOEs. The parameterization of the generics and default relationships between them, allowing more precise specifications on one hand and direct support for the developer on the other, seem to be new ideas, still requiring verification and optimization.

Generics have parameters that can be left uncompleted, meaning “any of ”, or completed, using other generic assignment to this parameter, similarly to the operations on the CC components. Generics can be refined or derived from others to meet developers’ specific needs. The generic domain concept is compliant with different security product types described in informative annexes C to F in [60].



TOE security environment - (Select by domain of application)

Generic:	Mnemonic description:	Relationships:
TDA.AuthInvDisg	Disguise [SAU] by an unauthorized [SNA] using user's invalid authentication	OACC.DataOperPrivCtrl, OIDA.AuthRetryLir
TDA.CmdDataAttack	Data [DAD] is referred to, inferred, modified, deleted, or added by an unauth	OACC.DataOperPrivCtrl, OACC.SysResCtrl,
TDA.ContTampDestroy	Information contents [DAD] are tampered or destroyed by an unauthorized [S	OACC.ContentPrivCtrl, OACC.ContentCtrl, C
TDA.CrpAnal	Card attacker [SNA] may compromise user data [DAD] being encrypted by th	OCON.BlockCipher, OEIT.StrongKey, OEIT
TDA.CrpAnal.D1	Intruz [SNA] może zdobyć dane użytkownika [DAD] zaszyfrowane przez TOE	
TDA.CrpAnal.D2	Intruz (CardIntruder) może zdobyć dane użytkownika (PlainText) zaszyfrowa	
TDA.CryptoResMod	Resources responsible for cryptographic operations, can be modified by auth	OEIT.TOEIntegrity, OIDA.MinPrivilUser, OID
TDA.CryptoTheft	Crypto analysis for theft of information. A hacker performs crypto analysis on	OCON.EmanCtrl, OCON.EncrypTextProt, O

TDA.CryptoResMod | CryptoResMod | TDA - Direct attacks, made by hackers and other intr

Basic | CRP - Cryptography

(History description, if version not basic)

Zasoby odpowiedzialne za wykonywanie operacji kryptograficznych mogą zostać zmodyfikowane przez podmioty autoryzowane [SAU] lub nieautoryzowane [SNA], osłabiając działanie operacji kryptograficznych

Resources responsible for cryptographic operations, can be modified by authorized [SAU] or unauthorized [SNA] subjects that will make these operations weaker

Proposed security objectives

Security objective:	Description:
OEIT.TOEIntegrity	TOE environment will provide T
OIDA.MinPrivilUser	Assigning the minimum privilege
OIDA.MultiLayAuth	Implementing multiple layers au
OIDA.LogonCtrl	The TOE will identify all users,
OIDA.IdentInfCtrl	Control of identification informat
OAVB.ServerIdle	Server management - safeguar
OAVB.StopProcUser	Stopping data processing by th
OSMN.DataBackup	Input data recovery from backu

C:\Documents and Settings\abialas\Dane aplikacji\SecCert\Library.xml

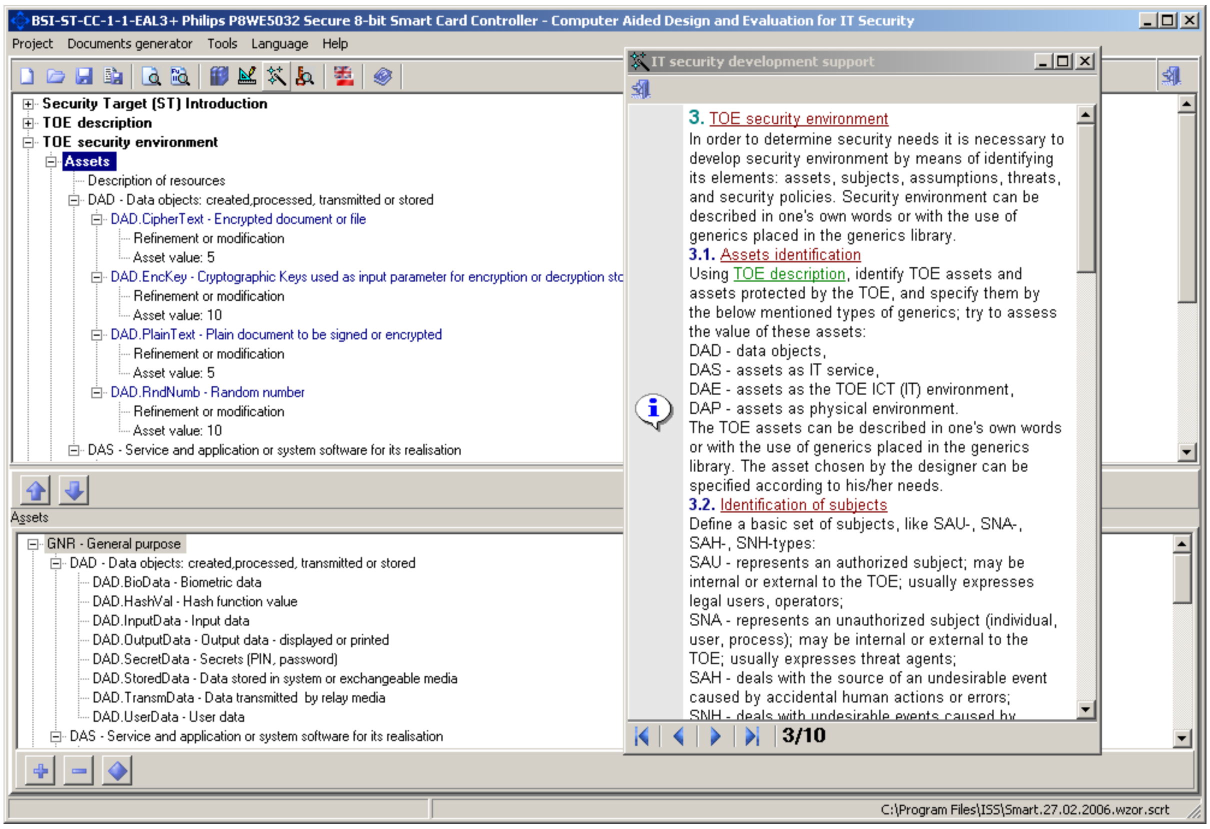
Fig. 12.1. ITSDF-tool – generic and component library window – example
Rys. 12.1. Przykład okienka aplikacji ITSDF-tool – biblioteka generyków i komponentów

All generics, including user-defined, and Common Criteria components – functional or assurance, are placed in the same program library (Fig. 12.1) which can be used for both the IT security development and its evaluation. All evaluation activities corresponding with the above basic elements are included in this library too. Developers can use all available generics and components existing in the library as building elements for their designs. They can add newly defined generics or even components as well.

The left part of the Fig.12.1 contains the library resources tree (i.e. specification means for the IT security development stages) while the right side presents some details concerning threats, and security objectives generics proposed for them. For the highlighted *TDA.CryptoResMod* threat its description field can be edited (in the Polish and English language versions) and the selected security objectives can be assigned as those proposed to cover this threat issue.

12.2. IT security development process support

The general scheme of the ST/PP development based on user requirements, PP specifications, or both, is wizard-driven and supported by the XML documents generator. The scheme was implemented, ensuring significant flexibility of:


BSI-ST-CC-1-1-EAL3+ Philips P8WE5032 Secure 8-bit Smart Card Controller - Computer Aided Design and Evaluation for IT Security
Project Documents generator Tools Language Help
IT security development support
Assets
3/10
C:\Program Files\ISS\Smart.27.02.2006.wzor.scrt

Security Target (ST) Introduction
TOE description
TOE security environment
Assets
Description of resources
DAD - Data objects: created,processed, transmitted or stored
DAD.CipherText - Encrypted document or file
Refinement or modification
Asset value: 5
DAD.EncKey - Cryptographic Keys used as input parameter for encryption or decryption stc
Refinement or modification
Asset value: 10
DAD.PlainText - Plain document to be signed or encrypted
Refinement or modification
Asset value: 5
DAD.RndNumb - Random number
Refinement or modification
Asset value: 10
DAS - Service and application or system software for its realisation

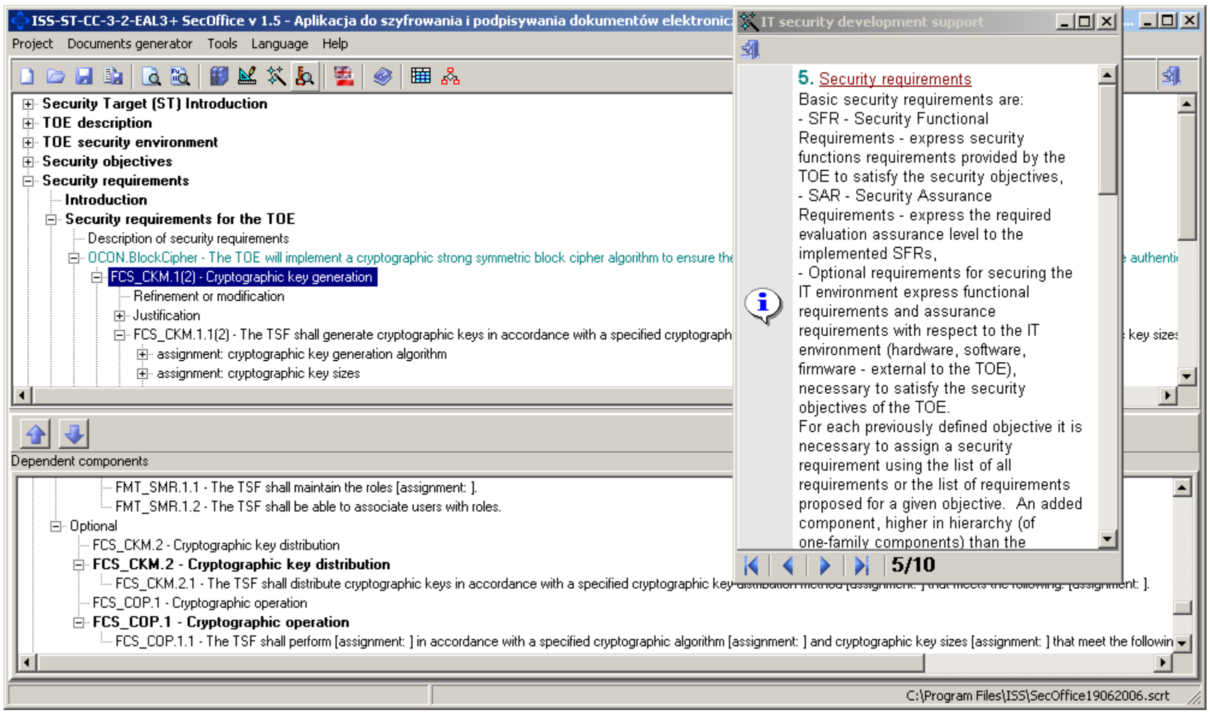
GNR - General purpose
DAD - Data objects: created,processed, transmitted or stored
DAD.BioData - Biometric data
DAD.HashVal - Hash function value
DAD.InputData - Input data
DAD.OutputData - Output data - displayed or printed
DAD.SecretData - Secrets (PIN, password)
DAD.StoredData - Data stored in system or exchangeable media
DAD.TransmData - Data transmitted by relay media
DAD.UserData - User data
DAS - Service and application or system software for its realisation

3. TOE security environment
In order to determine security needs it is necessary to develop security environment by means of identifying its elements: assets, subjects, assumptions, threats, and security policies. Security environment can be described in one's own words or with the use of generics placed in the generics library.
3.1. Assets identification
Using TOE description, identify TOE assets and assets protected by the TOE, and specify them by the below mentioned types of generics; try to assess the value of these assets:
DAD - data objects,
DAS - assets as IT service,
DAE - assets as the TOE ICT (IT) environment,
DAP - assets as physical environment.
The TOE assets can be described in one's own words or with the use of generics placed in the generics library. The asset chosen by the designer can be specified according to his/her needs.
3.2. Identification of subjects
Define a basic set of subjects, like SAU-, SNA-, SAH-, SNH-types:
SAU - represents an authorized subject; may be internal or external to the TOE; usually expresses legal users, operators;
SNA - represents an unauthorized subject (individual, user, process); may be internal or external to the TOE; usually expresses threat agents;
SAH - deals with the source of an undesirable event caused by accidental human actions or errors;
SNH - deals with undesirable events caused by

Fig. 12.2. ITSDF-tool – assets specification elaboration for the security environment
Rys. 12.2. Tworzenie specyfikacji zasobów dla otoczenia zabezpieczeń za pomoca narzędzia ITSDF-tool

- the security environment specification (Fig. 12.2), supporting trade-off between its elements, mostly between threats and OSPs,
- the security objectives specification, supporting trade-off between objectives declared for the TOE, for its environment or for both,
- the security requirements specification (Fig. 12.3), (including the requirements for the TOE environment) using CC components or RE-type generics and also different ways of assurance requirements selections.

In the Fig. 12.2 the assets specification elaboration for the security environment is presented. The upper window shows the specified assets with their values expressed with the use of measures predefined for the project while the lower window – the library window with library resources. The developer can move the selected item from the library to the to project, assign the value to the item, and add the refinement. On the right side there is the wizard window whose contents follows the designer’s activities.


```
ISS-ST-CC-3-2-EAL3+ SecOffice v 1.5 - Aplikacja do szyfrowania i podpisywania dokumentów elektronic
Project   Documents generator   Tools   Language   Help

+ Security Target (ST) Introduction
+ TOE description
+ TOE security environment
+ Security objectives
- Security requirements
    Introduction
  - Security requirements for the TOE
      Description of security requirements
    - OCON.BlockCipher - The TOE will implement a cryptographic strong symmetric block cipher algorithm to ensure the ... authenti
      - FCS_CKM.1(2) - Cryptographic key generation
          Refinement or modification
        + Justification
        - FCS_CKM.1.1(2) - The TSF shall generate cryptographic keys in accordance with a specified cryptograph ... key sizes
          + assignment: cryptographic key generation algorithm
          + assignment: cryptographic key sizes

Dependent components
        FMT_SMR.1.1 - The TSF shall maintain the roles [assignment: ].
        FMT_SMR.1.2 - The TSF shall be able to associate users with roles.
  - Optional
      FCS_CKM.2 - Cryptographic key distribution
    - FCS_CKM.2 - Cryptographic key distribution
        FCS_CKM.2.1 - The TSF shall distribute cryptographic keys in accordance with a specified cryptographic key ... nt: ].
      FCS_COP.1 - Cryptographic operation
    - FCS_COP.1 - Cryptographic operation
        FCS_COP.1.1 - The TSF shall perform [assignment: ] in accordance with a specified cryptographic algorithm [assignment: ] and cryptographic key sizes [assignment: ] that meet the followin

C:\Program Files\ISS\SecOffice19062006.scrt
```

```
IT security development support

5. Security requirements
Basic security requirements are:
- SFR - Security Functional
Requirements - express security
functions requirements provided by the
TOE to satisfy the security objectives,
- SAR - Security Assurance
Requirements - express the required
evaluation assurance level to the
implemented SFRs,
- Optional requirements for securing the
IT environment express functional
requirements and assurance
requirements with respect to the IT
environment (hardware, software,
firmware - external to the TOE),
necessary to satisfy the security
objectives of the TOE.
For each previously defined objective it is
necessary to assign a security
requirement using the list of all
requirements or the list of requirements
proposed for a given objective.  An added
component, higher in hierarchy (of
one-family components) than the
5/10
```

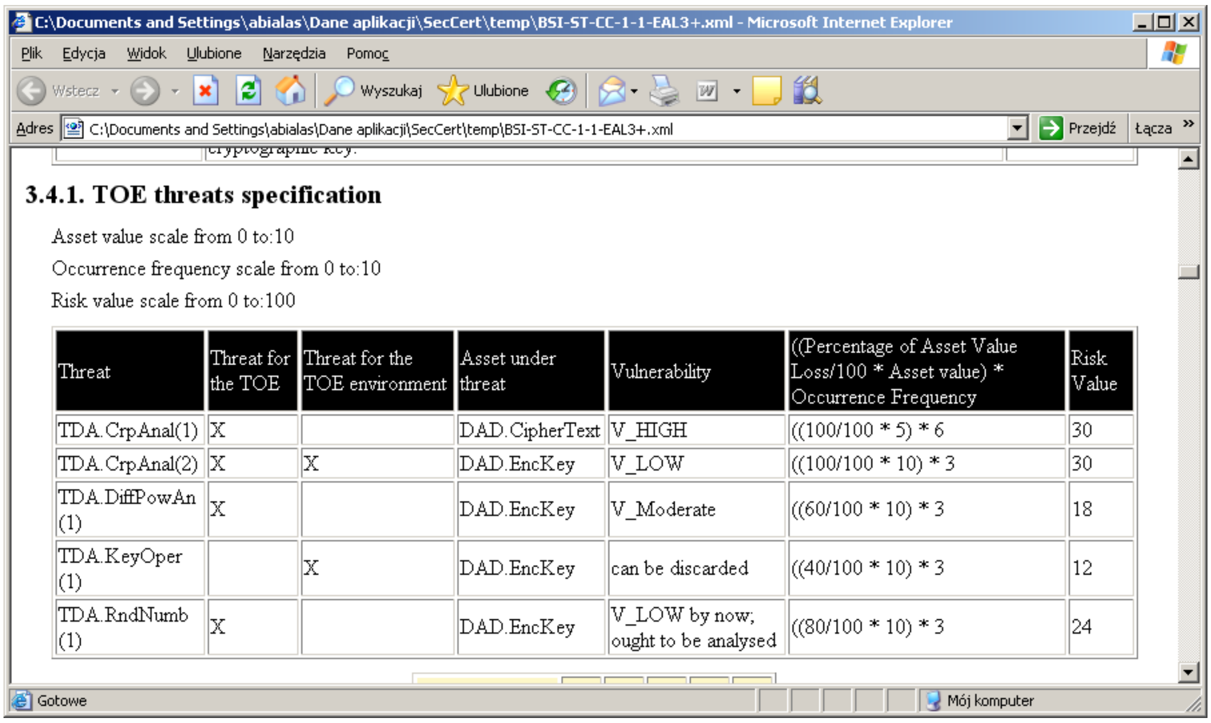
Fig. 12.3. ITSDF-tool – security functional requirements for the TOE selection
Rys. 12.3. Wybór wymagań funkcjonalnych dla przedmiotu oceny za pomocą narzędzia ITSDF-tool

The Fig. 12.3 shows the selection of the security functional requirements with the use of the ITSDF-tool. For a given security objective generic (e.g. *OCON.BlokCipher*, shown in the upper window) the security requirements (“proposed” or not) are assigned and justified. The lower window presents all CC-defined dependencies for the considered SFR. These dependencies are analyzed and added to the principal SFR by the developer, if needed. On the right side, the previously mentioned wizard is shown.

A simple built-in risk analyzer supporting developers in countermeasures selection, evaluation status/progress statistics and enhanced two-stage rationale (justification, rationale), with graphical presentations of all relationships, can be also very useful in the design and evaluation processes.

The ITSDF-tool has the simple risk analyzer built in. The part of the ST report generated as a result of the analysis is shown in the Fig. 12.4. The following elements were presented: assets, their values and risk calculation formula, and whether a given threat affects the TOE, its environment, or both. Risk value is an important factor in the security objectives-, and later the security functional requirements selection, as well as in defining the SOF claims (if applied).

At the end of the development process it should be demonstrated that the conformable TOE provided by its countermeasures will be secure in its environment. For these reasons, all ICT security development stages must be justified. Lists of corresponding items or graphical symbols (coloured rectangles and links), showing relations among model elements, are used (Fig. 12.5). During the rationale process a part of or the entire design can be visualized in a graphical or tabularized way. A developer, selecting a given item, can see all its relationships, influenced by the parameterization or mapping associations. More examples of the visualization can be found in the Appendix E.


C:\Documents and Settings\abialas\Dane aplikacji\SecCert\temp\BSI-ST-CC-1-1-EAL3+.xml - Microsoft Internet Explorer
Plik Edycja Widok Ulubione Narzędzia Pomoc
Wstecz Wyszukaj Ulubione
Adres C:\Documents and Settings\abialas\Dane aplikacji\SecCert\temp\BSI-ST-CC-1-1-EAL3+.xml
Przejdź łącza
Gotowe
Mój komputer

3.4.1. TOE threats specification

Asset value scale from 0 to:10

Occurrence frequency scale from 0 to:10

Risk value scale from 0 to:100

Threat	Threat for the TOE	Threat for the TOE environment	Asset under threat	Vulnerability	((Percentage of Asset Value Loss/100 * Asset value) * Occurrence Frequency	Risk Value
TDA.CrpAnal(1)	X		DAD.CipherText	V_HIGH	((100/100 * 5) * 6	30
TDA.CrpAnal(2)	X	X	DAD.EncKey	V_LOW	((100/100 * 10) * 3	30
TDA.DiffPowAn (1)	X		DAD.EncKey	V_Moderate	((60/100 * 10) * 3	18
TDA.KeyOper (1)		X	DAD.EncKey	can be discarded	((40/100 * 10) * 3	12
TDA.RndNumb (1)	X		DAD.EncKey	V_LOW by now; ought to be analysed	((80/100 * 10) * 3	24

Fig. 12.4. Simple risk analyzer – results presented in the ST report
Rys. 12.4. Prosty analizator ryzyka – wyniki analizy z raportu zadania zabezpieczeń

Finally, the developer will be able to create automatically ST or PP documents, using all sampled and verified data. An example dealing with the mentioned SecOffice Security Target was shown in the Fig. 12.6. The SecOffice is a cryptographic application (encryption and digital signature). The figure presents its TOE boundaries on a picture attached by the ITSDF-tool. Please note that graphical data (schemes) can be attached as well. The ITSDF-tool allows to issue two kinds of such reports: standard (CC-defined PP or ST) or their extended versions which contain additional data used for their elaboration.The whole IT security development process is wizard-driven, which was shown on the above Fig 12.2 and Fig 12.3.

To sum up, the prototype of the tool meets basic needs of an IT security developer:

- precisely expresses security design needs, allowing to build a security model for the TOE,

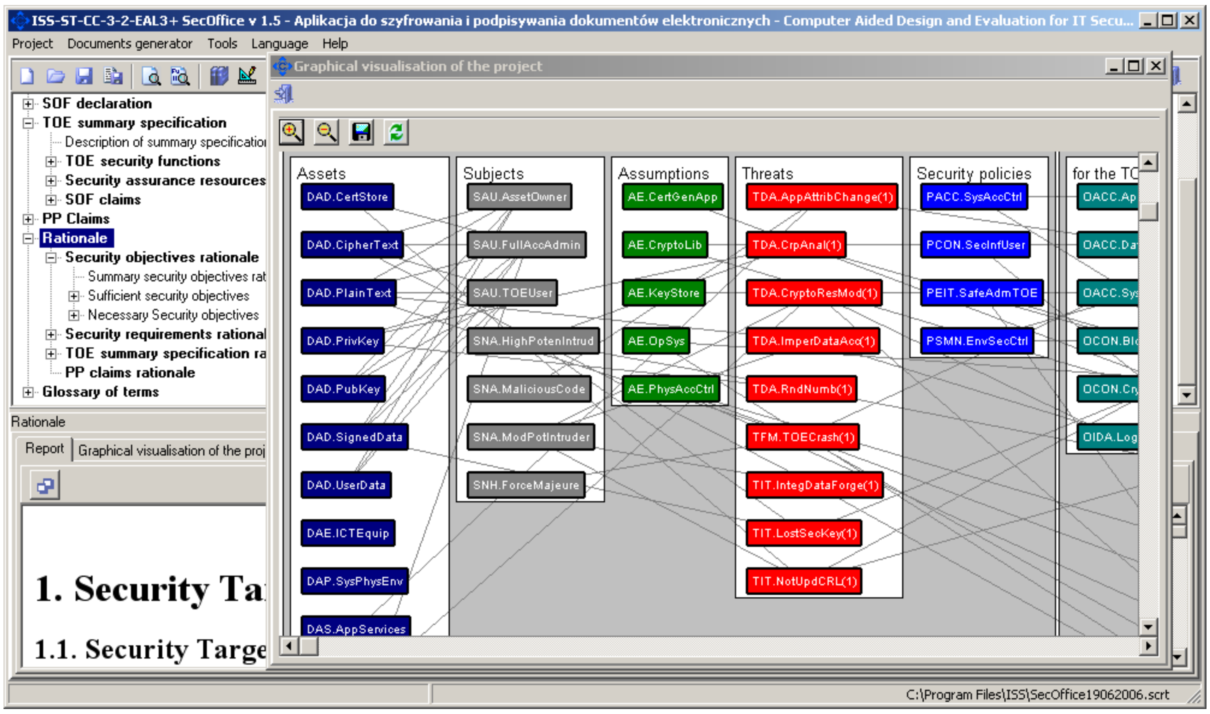
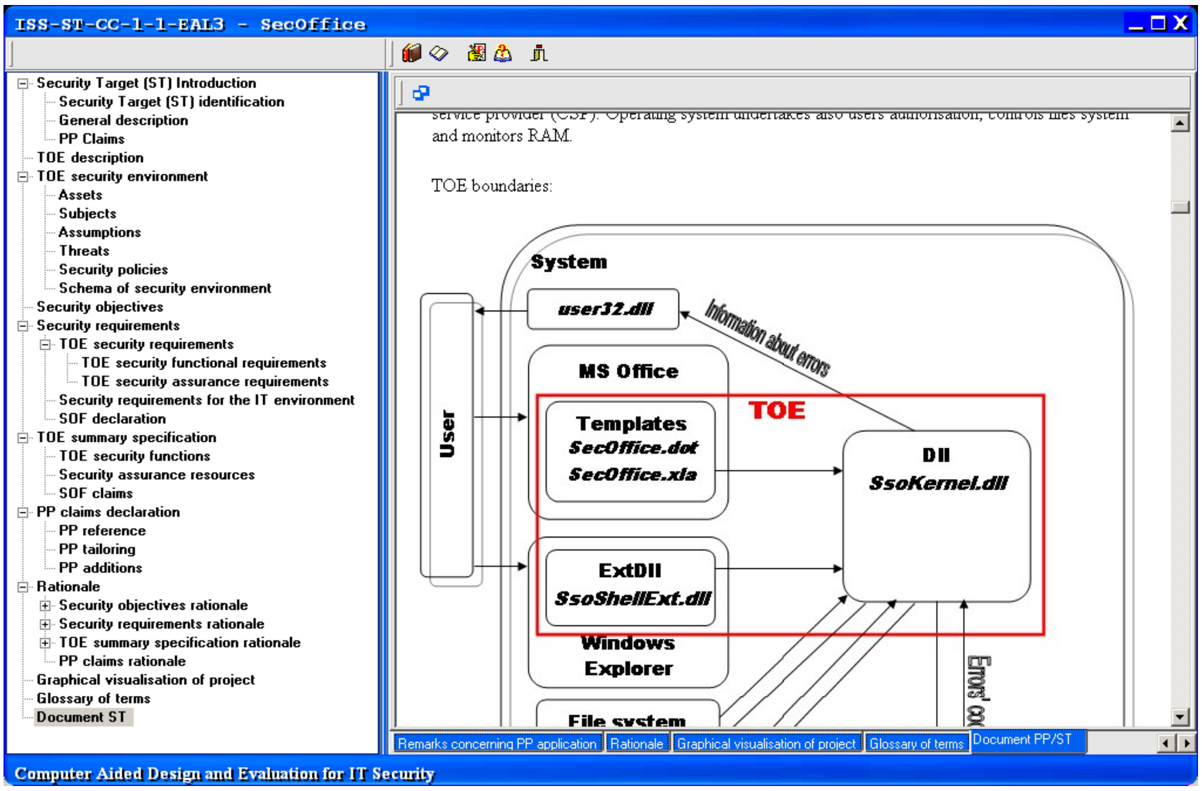


Fig. 12.5. Graphically supported ST rationale
Rys. 12.5. Graficzne wspomaganie procesu uzasadnienia zadania zabezpieczeń

- supports the security model refinement process – starting from ideas and needs, through risk assessment to security functions specification, and delivering and managing assurance measures,
- supports obligatory rationale processes between each of the design stages (temporary reports, graphical presentation of the relations),
- facilitates documentation creating and management,
- automatically generates the ST and PP documents,
- supports reusability of elaborated security models.


ISS-ST-CC-1-1-EAL3 - SecOffice

- Security Target (ST) Introduction
  - Security Target (ST) identification
  - General description
  - PP Claims
- TOE description
- TOE security environment
  - Assets
  - Subjects
  - Assumptions
  - Threats
  - Security policies
  - Schema of security environment
- Security objectives
- Security requirements
  - TOE security requirements
    - TOE security functional requirements
    - TOE security assurance requirements
  - Security requirements for the IT environment
  - SOF declaration
- TOE summary specification
  - TOE security functions
  - Security assurance resources
  - SOF claims
- PP claims declaration
  - PP reference
  - PP tailoring
  - PP additions
- Rationale
  - Security objectives rationale
  - Security requirements rationale
  - TOE summary specification rationale
  - PP claims rationale
- Graphical visualisation of project
- Glossary of terms
- Document ST

service provider (CSP). Operating system undertakes also users authorisation, controls files system and monitors RAM.

TOE boundaries:

Remarks concerning PP application | Rationale | Graphical visualisation of project | Glossary of terms | Document PP/ST

Computer Aided Design and Evaluation for IT Security

Fig. 12.6. Automatically generated ST report
Rys. 12.6. Automatyczne tworzenie specyfikacji zadania zabezpieczeń (ST)

12.3. IT security evaluation support

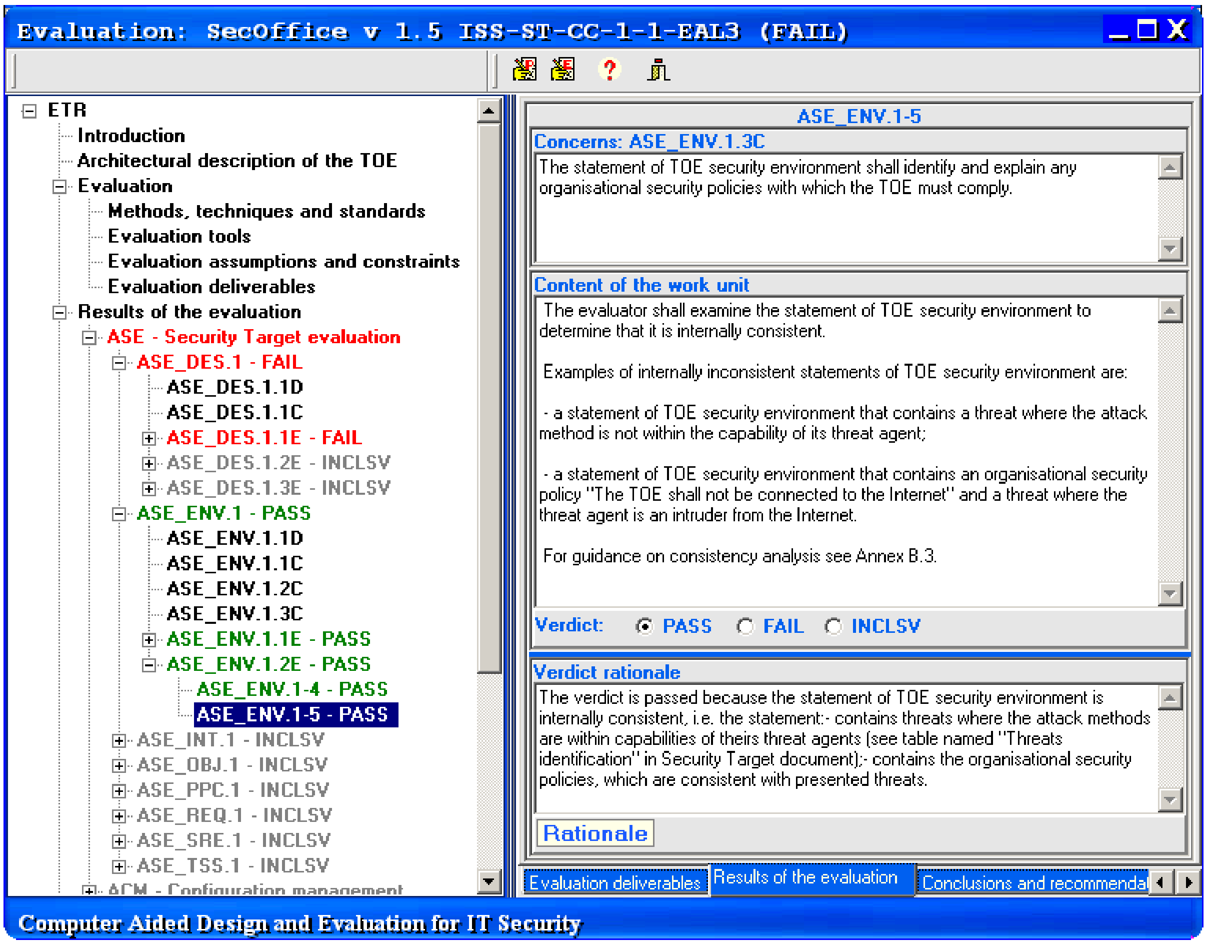
The presented tool allows to aid the whole self-evaluation process for Security Targets, Protection Profiles, and the TOE vs. the claimed EAL. In short, the ICT security evaluation process corresponds with elaborating the ETR. In the Fig. 12.7 the evaluator's application window was presented. On the left side the created ETR structure is shown. The evaluator checks if the ST was properly elaborated (ASE class), particularly she/he checks if the ST meets the ASE_ENV.1 component requirement concerning the security environment workout. Please note the ASE_ENV.1-5 work unit (an elementary action of the evaluator) concerning the ASE_ENV.1.3C content and presentation element, shown on the right side. The evaluator assigns a verdict and issues its rationale. More details concerning the evaluation process are included in the [45-46].

The ETR includes an introduction, the TOE description and different topics concerning the evaluation process, including "evaluation deliverables" called evidence. Evidence documentation influenced by the assurance requirements and attached by the developers to the ST/PP are managed with the use of the ITSDF-tool.

The main part of the ETR, called „Results of evaluation”, has APE or ASE class components attached and all components contained in the declared EAL package with its optional augmentation.

It is obvious that each assurance component has three basic elements implemented, used in the evaluation process [45-46], i.e. D, C, E elements. The evaluator can review the elements of each component and corresponding evaluation activities with work units. All work units should be reviewed against the evidence material delivered by the developer, and verdicts should be assigned with the evaluator’s obligatory justification. The verdicts are accumulated, allowing to monitor the evaluation progress and current results (not shown there).

The ETR and other reports could be automatically generated (all XML-based) at any stage of the evaluation process. Built-in statistics show evaluation extent and progress.



Evaluation: SecOffice v 1.5 ISS-ST-CC-1-1-EAL3 (FAIL)

```
ETR
  Introduction
  Architectural description of the TOE
  Evaluation
    Methods, techniques and standards
    Evaluation tools
    Evaluation assumptions and constraints
    Evaluation deliverables
  Results of the evaluation
    ASE - Security Target evaluation
      ASE_DES.1 - FAIL
        ASE_DES.1.1D
        ASE_DES.1.1C
        ASE_DES.1.1E - FAIL
        ASE_DES.1.2E - INCLSV
        ASE_DES.1.3E - INCLSV
      ASE_ENV.1 - PASS
        ASE_ENV.1.1D
        ASE_ENV.1.1C
        ASE_ENV.1.2C
        ASE_ENV.1.3C
        ASE_ENV.1.1E - PASS
        ASE_ENV.1.2E - PASS
          ASE_ENV.1-4 - PASS
          ASE_ENV.1-5 - PASS
      ASE_INT.1 - INCLSV
      ASE_OBJ.1 - INCLSV
      ASE_PPC.1 - INCLSV
      ASE_REQ.1 - INCLSV
      ASE_SRE.1 - INCLSV
      ASE_TSS.1 - INCLSV
    ACM - Configuration management
```

ASE_ENV.1-5

Concerns: ASE_ENV.1.3C

The statement of TOE security environment shall identify and explain any organisational security policies with which the TOE must comply.

Content of the work unit

The evaluator shall examine the statement of TOE security environment to determine that it is internally consistent.

Examples of internally inconsistent statements of TOE security environment are:

- a statement of TOE security environment that contains a threat where the attack method is not within the capability of its threat agent;

- a statement of TOE security environment that contains an organisational security policy "The TOE shall not be connected to the Internet" and a threat where the threat agent is an intruder from the Internet.

For guidance on consistency analysis see Annex B.3.

Verdict: [✓ PASS] FAIL INCLSV

Verdict rationale

The verdict is passed because the statement of TOE security environment is internally consistent, i.e. the statement:- contains threats where the attack methods are within capabilities of theirs threat agents (see table named "Threats identification" in Security Target document);- contains the organisational security policies, which are consistent with presented threats.

Rationale
Evaluation deliverables
Results of the evaluation
Conclusions and recommenda
Computer Aided Design and Evaluation for IT Security

Fig. 12.7. Evaluator's application window of the ITSDF-tool
Rys. 12.7. ITSDF-tool jako narzędzie do prowadzenia oceny zabezpieczeń

To sum up, the tool meets the IT security evaluator's basic expectations:

- supporting step by step the arduous evaluation process,
- facilitating the creation and management of the documentation, including evidence material,
- allowing on-line monitoring of the progress and the results of the evaluation,
- automatically generating the Evaluation Technical Reports (ETR), Observation Reports (OR), and the Evaluation Discovery Report (EDR),
- allowing to master many details and relationships between assurance components and evidence material during the self-evaluation process.

13. CONCLUSIONS

The monograph presents the concept of the UML framework for Common Criteria developers and the computer-aided tool for the IT security products development, created on this basis. By modelling the IT security development process with respect to the security-related products or systems, the following objectives can be achieved:

- a simplified picture of such complex products is created, containing their structural and behavioural specifications,
- these specifications can be analyzed to gain knowledge on these products, their evaluation, usage and improvements,
- the sampled knowledge is reusable,
- the created models document developers' decisions on-line.

The IT Security Development Framework and the related methodology are open to the currently developed concepts, like:

- security engineering principles (e.g. separation of concerns),
- using the UML for IT security as the application domain (e.g. UMLsec),

and these approaches support each other.

The work presented in the monograph encompasses the following:

- identifications of gaps of the existing methodologies and tools – preliminary studies and researches concerning the Common Criteria family of standards implementation, selected Security Targets and Protection Profiles [87-88], case studies using one's own developed IT product [95], reviewing available tools, studying best practices, publications, research papers, discussing the topic with experts [5-7], [10], [12-15], [19], [22], [24], [26-30], [32-33];
- identification of the developers' needs – analyzing development processes, identifying points whose support for the developers is especially required for better preciseness and lower cost, like:
  - selection of the right security items to cover other items (e.g. security objectives to cover threats, OSPs and assumptions, requirements to cover objectives, etc.);
  - selection of the assurance requirements from different sources and merging them;

  - enhanced SOF claims elaboration scheme;
  - facilitating the rationale and risk analysis processes;
- elaborating the enhanced IT Security Development Framework, UML-based, and CC-compliant;
  - development of the IT security specification language as the set of enhanced generics, allowing mapping, parameterization and operations;
  - working out a design library on this basis, containing Common Criteria components and a set of enhanced generics; defining a basic set of relations between the generics to better support the developers in covering security items (threats by security objectives, objectives by components, etc.), semiformally defined generics were implemented as the library;
  - elaborating general class diagrams for the Security Targets and Protection Profiles, and their detailed sub-diagrams for every development stage, making reference to all main and auxiliary data and activities of the development process;
  - elaborating a set of detailed activity diagrams describing the IT security development process; the diagrams are implemented as the wizard supporting the developers step-by-step;
  - satisfying the assumptions specified at end of the section 2.2, i.e.: compatibility with the ISO/IEC TR 15446:2004 standard, enhanced 2-step rationale, a simple risk analyzer built in the tool, (self-)evaluation facility, etc;
  - supplementing the semiformal framework with the formal elements to reach better modelling preciseness;
- the incremental development of the tool prototype based on feedbacks from case studies and experimentation provided by the security lab and trainings (using the existing security targets and protection profiles, and the developed ones – digital signature application, smart card system); validation on the COTS-type digital signature/encryption application and on the existing security targets and protection profiles examples.

The presented framework focuses on the use cases, specifying expected behaviour of the security-related product or system, and the threats influencing them negatively. It is architecture-centric and independent of the developed product, while the architecture is defined by the Common Criteria standard, and ST and PP are the basic artefacts. It allows an iterative, recursive and risk-driven CC development approach, based on the continuous design improvement and rationale support.

The following features help to achieve the assurance for the IT product or system in a more efficient way:

- issuing detailed activity diagrams, implemented as the tool wizard – step-by-step support given to developers during the IT security development process; the trade-off between the TOE and its environment facility is especially useful to issue the final specification of security objectives;
- issuing specifications that are more precise and coherent than those in [60] by providing the developers with specification means comparable with CC components at any development stage; generics are semiformal and flexible means (due to their: parameterization, iteration, refinement); developers are provided with a rich (about 400 items) but open library of generics for typical security issues; developers can define their own generics on the condition that they use a predefined format; building generics and/or components chains being solutions to elementary security problems;
- reaching design reusability in the same way as for other computer-aided systems;
- better decision support for developers (default relationships, i.e. "proposed", easy checking of variants, the trade-off between the TOE and its environment, the trade-off between technical and organizational measures);
- supporting risk analysis; as the specified security objectives are formulated on the basis of risk value, they are more adequate;
- enhanced SOF-claims management (defining and implementation checking);
- merging the assurance requirements of different origin;
- extending the OSPs allows to achieve better compliance with the information security management standards; the generics library is provided by the set of many OSPs, including those compatible with the information security management standards; it is easier to incorporate the developed and evaluated TOE in the operational environment and its security management system;
- improving documentation management, including the preparation of evidence, reporting, statistics, etc;
- allowing preliminary self-evaluation of the work, similarly to the evaluation process performed by the security labs;
- assisting the rationale process by means of the visualization tool;
- and, finally, automatically issuing different kind of reports, including ST or PP reports, full-design reports, self-evaluation reports.

It can be seen that the tool makes the IT security development process easier, especially the rationale between development stages, supported by the visualization facility. Generally, the formalization of the IT security development process allows to issue more precise and concise specifications. The "language of generics" is more precise and compact than the description of informal security features and behaviours, and it is better understood by the developers than some formal methods.

The main contribution of the monograph is:

- the concept and implementation of the CC-compliant and UML-based IT Security Development Framework,
- enhanced language for elementary security issues specification,
- the scheme of the security-related product functionality capture (i.e. workout of the TOE description), based on a UML use case diagram and collaborations, considering not only legal but also illegal users and sources of undesirable events,
- security environment elaboration based on the risk analysis,
- 2-step enhanced rationale (any item justification, CC rationale for each development stage),
- the concept of mapping the many-to-many relationships of the threats, policies, and requirements for the system – supporting chains of the elementary security issues.

It was shown that:

- using the UML approach enables to express the IT security development process and the IT security features of the designed products as well,
- using the UML specification method, especially use cases, makes it possible to describe security-related products more precisely in TOE description (leading part of PP/ST specification), allowing to better capture their security features and input on the development process,
- using the UML security-related products specification and risk analysis makes the developed security environment more coherent and more consistent with other parts of the PP/ST specification,
- defining the use of UML domain-oriented subsets of generics makes it easier and more effective to drive threats and policies to the security objectives and, later, the requirements and security functions,
- introducing different views provides proper information on the design for different Common Criteria consumers, like: IT security developers, evaluators, users, administrators, and sponsors.

Please note that most of the diagrams presented there cannot be directly transferred to the code. Many of them are activity diagrams performed by people (i.e. developers) not by the computer software. There is a deep analogy between the modelling of IT security development processes and enterprise process modelling, the latter being the basic domain of the UML applications.

On the basis of the achieved results the following tasks are planned:

- educating the CC community to better understand the role of the UML-based modelling in the IT security development process,
- disseminating the UML modelling knowledge and expertise in the CC community,
- supporting the deployment of the Common Criteria in Poland.

The results achieved up until now indicate that design and evaluation processes seem to be considerably facilitated but the tool needs more verification in real operation and more user feedbacks. The presented supporting ITSDF-tool prototype has been continuously improved. Some useful features, like: packages management, composite and complex TOE development, better reporting and better projects management are under development.

The correct ST/PP structure and content and all TOE specification elements are enforced by the tool, helping to avoid most of the CC developers' problems [2].

The methodology presented there was developed to meet the general hypothesis concerning the basis of the assurance. This hypothesis says that the more rigorous the developed IT product or system is, the more assurance it has. More rigorous means more formal, going from informal approach for the lowest EALs, through semiformal for the middle, and to the highest EALs allowing formal methods only.

The methodology presented there is generally semiformal, but it has some facilities or options having formal character, such as:

- the possibility to adopt on entry the IT product or system models having formal character, e.g. from [72],
- the use of the OCL which supports precise modelling,
- the possibility to create better formalized security models, based on the introduced facilities (generics, semantics, ontology [106]).

There are some limitations to the formalizations, i.e. the cost and time of the development process. The formal methods are rather hard to use, they are focused on chosen problems only, and are not very popular among engineers. For these reasons, the cost/benefit analysis is recommended, performed usually at the beginning of the development process by the IT product or system sponsor. Only adequate methods and means can be used for products of given features and applications. Thus the sets of different methods ant tools are needed. The methodology presented there is one of them. It provides the set of distinguished means for different applications and required assurance, and should be open to incorporate other methods designed for specific applications.

Going towards the design for the securability idea, the main objective of this work is to create a CC-compliant and UML-based IT Security Development Framework, supporting the use of security engineering principles through the whole IT security development process.

Creating better frameworks and related tools means more effective projects and shortening developers' learning curves, which is especially important for COTS developers.

The issues presented in the monograph are very extensive and focused only on the developed ITSDF framework and its implementation. For this reason the monograph cannot be considered as a kind of guidelines to the Common Criteria methodology or a discussion of its current problems or challenges. The monograph does not discuss the advantages or disadvantages of this methodology and current problems to solve either.

One of the reviewers refers to the Common Criteria shortcomings, asking how they are compensated by the developed framework and tool. Let us discuss some of them.

1. The time consuming IT security development and evaluation processes concern the given version of the IT product or system. Moreover, when a new version is created, recertification is needed.

The ITSDF framework (and especially the ITSDF-tool) provides solution to this issue by improving the project reusability.

2. The assurance of the IT system composed from the evaluated IT product or system was difficult to asses directly.

This concerns the older versions of the CC standard. Starting from the CC v3.x the composability is supported and it is much more easier to solve this problem, still, the current version of the ITSDF-tool has not implemented this feature so far. Currently, a new version of the tool is developed and it does have this feature.

3. The CC methodology is focused on static aspects of the system behaviour.

This can be compensated by formal methods and the UML-based models expressing dynamic aspects of systems, like UMLsec. The monograph shows how to integrate the ITSDF and UMLsec approaches.

4. There is a need to integrate the CC methodology with the information security management methodology, especially to harmonize security policy rules.

The ITSDF framework supports the solution of this issue by providing a basic set of policy rules (and the possibility to define others), compliant with the ISO/IEC 27001 (see Appendix E/Example E.7 and [12], [33]).

Each year the specialists (CC authors, practitioners, R&D performers, government bodies, evaluators, sponsors, etc.) discuss the current state and new challenges of the CC world, meeting together at the International Common Criteria Conference (ICCC) [42]. During the last one, the 8th ICCC held in Rome, the general concept of the ITSDF framework was presented.

The Common Criteria methodology [42], recognized as a matured one, still has some areas that need research and development to reach better effectiveness and friendliness, or new areas of applications.

APPENDIX A. BASIC COMMON CRITERIA TERMINOLOGY

This appendix includes the basic Common Criteria terminology [38] to make the reading of this work easier.

Table A.1

Basic terminology used in the monograph

Assets	Information or resources to be protected by the countermeasures of the TOE.
Assignment	The specification of an identified parameter in a component.
Assurance	Grounds for confidence that an entity meets its security objectives.
Attack potential	The perceived potential for success of an attack, should the attack be launched, expressed in terms of an attacker's expertise, resources and motivation.
Augmentation	The addition of one or more assurance component(s) from CC Part 3 to an EAL or assurance package.
Authentication data	Information used to verify the claimed identity of a user.
Authorized user	A user who may, in accordance with the TSP, perform an operation.
Class	A grouping of families that share a common focus.
Component	The smallest selectable set of elements that may be included in a PP, an ST, or a package.
Connectivity	The property of the TOE which allows interaction with IT entities external to the TOE. This includes exchange of data by wire or by wireless means over any distance in any environment or configuration.
Dependency	Relationship between requirements where the dependent requirements must normally be satisfied for other requirements to be able to meet their objectives.
Element	An indivisible security requirement.
Evaluation	Assessment of a PP, ST or TOE against defined criteria.
Evaluation Assurance Level (EAL)	A package consisting of assurance components from Part 3 that represents a point on the CC predefined assurance scale.
Evaluation authority	A body that implements the CC for a specific community by means of an evaluation scheme and thereby sets the standards and monitors the quality of evaluations conducted by bodies within that community.
Evaluation scheme	The administrative and regulatory framework under which the

	CC is applied by an evaluation authority within a specific community.
Extension	The addition to an ST or PP of functional requirements not contained in CC Part 2 and/or assurance requirements not contained in Part 3 of the CC.
External IT entity	Any IT product or system, distrusted or trusted, outside the TOE, that interacts with the TOE.
Family (concerning components)	A grouping of components that share security objectives but may differ in emphasis or rigour.
Formal	Expressed in a restricted-syntax language with defined semantics based on well-established mathematical concepts.
Human user	Any person who interacts with the TOE.
Identity	A representation (e.g. a string) uniquely identifying an authorized user, which can either be the full or abbreviated name of that user or a pseudonym.
Informal	Expressed in natural language.
Internal communication channel	A communication channel between separated parts of the TOE.
Internal TOE transfer	Communicating data between separated parts of the TOE.
Inter-TSF transfers	Communicating data between the TOE and the security functions of other trusted IT products.
Iteration	The use of a component more than once with varying operations.
Object	An entity within the TSC that contains or receives information and upon which subjects perform operations.
Organizational security policies (OSP)	One or more security rules, procedures, practices, or guidelines imposed by an organization upon its operations.
Package	A reusable set of either functional or assurance components (e.g. an EAL), combined together to satisfy a set of identified security objectives.
Product	A package of IT software, firmware and/or hardware providing functionality designed for use or incorporation within a multiplicity of systems.
Protection Profile (PP)	An implementation-independent set of security requirements for a category of TOEs that meet specific consumer needs.
Reference monitor	The concept of an abstract machine that enforces TOE access control policies.
Reference validation mechanism	An implementation of the reference monitor concept that possesses the following properties: it is tamperproof, always invoked, and simple enough to be subjected to thorough analysis and testing.
Refinement	The addition of details to a component.
Role	A predefined set of rules establishing the allowed interactions between a user and the TOE.
Secret	Information that must be known only to authorized users and/or the TSF in order to enforce a specific SFP.

Security attribute	Information associated with subjects, users and/or objects that is used for the enforcement of the TSP.
Security Function (SF)	A part or parts of the TOE that have to be relied upon for enforcing a closely related subset of rules from the TSP.
Security Function Policy (SFP)	The security policy enforced by an SF.
Security objective	A statement of intent to counter identified threats and/or satisfy identified organization security policies and assumptions.
Security Target (ST)	A set of security requirements and specifications to be used as the basis for evaluation of an identified TOE.
Selection	The specification of one or more items from a list in a component.
Semiformal	Expressed in a restricted-syntax language with defined semantics.
Strength of Function (SOF)	A qualification of the TOE security function expressing the minimum efforts assumed necessary to defeat its expected security behaviour by directly attacking its underlying security mechanisms.
SOF-basic	A level of the TOE strength of function where analysis shows that the function provides adequate protection against a casual breach of TOE security by attackers possessing low attack potential.
SOF-medium	A level of the TOE strength of function where analysis shows that the function provides adequate protection against a straightforward or intentional breach of TOE security by attackers possessing moderate attack potential.
SOF-high	A level of the TOE strength of function where analysis shows that the function provides adequate protection against a deliberately planned or organized breach of TOE security by attackers possessing high attack potential.
Subject	An entity within the TSC that causes operations to be performed.
System	A specific IT installation, with a particular purpose and operational environment.
Target of Evaluation (TOE)	An IT product or system and its associated administrator and user guidance documentation that is the subject of evaluation.
TOE resource	Anything usable or consumable in the TOE.
TOE Security Functions (TSF)	A set consisting of all hardware, software and firmware of the TOE that must be relied upon for the correct enforcement of the TSP.
TOE Security Functions Interface (TSFI)	A set of interfaces, whether interactive (man-machine interface) or programmatic (application programming interface), through which TOE resources are accessed, mediated by the TSF, or information is obtained from the TSF.
TOE Security Policy (TSP)	A set of rules that regulate how assets are managed, protected and distributed within the TOE.
TOE security policy model	A structured representation of the security policy to be enforced by the TOE.

Transfers outside TSF control	Communicating data to entities not controlled by the TSF.
Trusted channel	A means by which a TSF and a remote trusted IT product can communicate with necessary confidence to support the TSP.
Trusted path	A means by which a user and a TSF can communicate with necessary confidence to support the TSP.
TSF data	Data created by and for the TOE that might affect the operation of the TOE.
TSF Scope of Control (TSC)	The set of interactions that can occur with or within the TOE and are subject to the rules of the TSP.
User	Any entity (human user or external IT entity) outside the TOE that interacts with the TOE.
User data	Data created by and for the user that do not affect the operation of the TSF.

APPENDIX B. OBJECT CONSTRAINT LANGUAGE (OCL) SYNTAX AND SEMANTICS – THE USED DEFINITIONS AND TERMS

This appendix, including the basic definitions and terms, was elaborated on the basis of the Appendix A of the OMG document [83]. Only those terms and definitions were selected that are used for the well-formedness of security models specification.

$\mathcal{A}$ – represents an alphabet;

$\mathcal{N}$ – a set of finite, non-empty names $\mathcal{N} \subseteq \mathcal{A}^+$ over the alphabet $\mathcal{A}$;

N – a set of non-negative integers;

$\mathbf{N}_n$ – a set of non-negative integers up to n and including n, for any $n \in N$;

$\mathcal{P}(\mathcal{X})$ – a set of subsets of a set $\mathcal{X}$;

$\sum$ – a signature $\sum=(\mathcal{T}, \Omega)$, where:

$\mathcal{T}$ – a set of type names,

Ω – a set of operations over types in $\mathcal{T}$;

$\perp$ – undefined (unknown, null) value added to every type domain;

$\mathcal{C}\text{LASS} \subseteq \mathcal{N}$ – the set of classes is a finite set of names; a class represents a common description for a set of objects having the same properties; each class induces an object type $t_c \in \mathcal{T}$, having the same name as the class;

$\mathcal{A}\text{TT}_c$ – the attributes of a class $c \in \mathcal{C}\text{LASS}$ are defined as the set $\mathcal{A}\text{TT}_c$ of signatures a: $t_c \rightarrow t$, where $t \in \mathcal{T}$ is a type, $t_c \in \mathcal{T}$ is a type of class c, and $a \in \mathcal{N}$ is the attribute name; attributes are part of a class declaration; it is assumed that an attribute name may not be used to define another attribute with a different type, i.e.:

$$\forall t, t' \in \mathcal{T}: ((a: t_c \rightarrow t \in \mathcal{A}\text{TT}_c) \wedge (a: t_c \rightarrow t' \in \mathcal{A}\text{TT}_c)) \Rightarrow t=t';$$

$\mathcal{O}\text{P}_c$ – the operations of a class $c \in \mathcal{C}\text{LASS}$ are defined by a set $\mathcal{O}\text{P}_c$ of signatures $\omega: t_c \times t_1 \times \ldots \times t_n \rightarrow t$, where $t, t_1, \ldots , t_n \in \mathcal{T}$ are types, $t_c \in \mathcal{T}$ is a type of class c and $\omega \in \mathcal{N}$ is the operation symbol; an operation may have any number of parameters, but only a single return type $t \in \mathcal{T}$.

$\mathcal{A}\text{SSOC}$ – the set of associations is defined by:

i. a finite set of names $\mathcal{A}\text{SSOC} \in \mathcal{N}$, and

ii. the function associates: $\mathrm{ASSOC} \rightarrow \mathrm{CLASS}^+$, as $\mapsto \langle c_1, \dots, c_n \rangle$, with ($n \geq 2$);

the function associates maps each association name as $\in$ ASSOC to a finite list of classes participating in the association; associations represent structural relationships between classes;

roles – role names for an association are defined by the function roles:

$\mathrm{ASSOC} \rightarrow \mathcal{N}^+$, as $\mapsto \langle r_1, \dots, r_n \rangle$, with ($n \geq 2$), where

$\forall i, j \in \{1, \dots, n\}: i \neq j \Rightarrow r_i \neq r_j$ (role names distinction);

the function roles(as) $= \langle r_1, \dots, r_n \rangle$ assigns a unique role name r_i to each class c_i for $i \in \{1, \dots, n\}$ participating in the association; the role names may be omitted; in this case the class name with its first letter changed to the lower case can be used as the role name;

participating – this function returns the set of associations the class participates in; participating:

$\mathrm{CLASS} \rightarrow P(\mathrm{ASSOC})$, $c \mapsto \{as \mid as \in \mathrm{ASSOC} \wedge \mathrm{associates}(as) = \langle c_1, \dots, c_n \rangle$

$\exists i \in \{1, \dots, n\}: c_i = c\}$;

navends – this function returns the set of all role names reachable or navigable from a class over a given association (the uniqueness of role names when a class is a part of many associations should be ensured); the function navends:

$\mathrm{CLASS} \times \mathrm{ASSOC} \rightarrow P(\mathcal{N})$,

$(c, as) \mapsto \{r \mid \mathrm{associates}(as) = \langle c_1, \dots, c_n \rangle \wedge \mathrm{roles}(as) = \langle r_1, \dots, r_n \rangle \wedge$

$\exists i,j \in \{1, \dots, n\}: (i \neq j \wedge c_i = c \wedge r_j = r)\}$;

navends(c) – this function returns the set of all role names reachable from a class c along all associations the class participates in; navends(c):

$\mathrm{CLASS} \rightarrow P(\mathcal{N})$,

$$c \mapsto \bigcup_{as \in \mathrm{participating}(c)} \mathrm{navends}(c, as);$$

multiplicities; assuming that as $\in$ ASSOC and associates(as) $= \langle c_1, \dots, c_n \rangle$, the function multiplicities(as) $= <\mathcal{M}_1, \dots, \mathcal{M}_n>$ assigns a non-empty set $\mathcal{M}_i \subseteq N_0$ with $\mathcal{M}_i \neq \{0\}$ to each class c_i participating in the association; the $\mathcal{M}_i$ represents a number of links and an object of the class c_i can be part of each link;

generalization hierarchy $\prec$ is a partial order on the set of classes, marked CLASS, expressing their taxonomy relationships;

child and **parent classes**; let us assume that the classes $c_1, c_2 \in$ CLASS with $c_1 \prec c_2$;

c_1 is called a child class of c_2, and c_2 is called a parent class of c_1;

parents(c):

$\mathrm{CLASS} \rightarrow \mathcal{P}(\mathrm{CLASS})$,

$c \mapsto \{c' \mid c' \in \mathrm{CLASS} \wedge c \prec c'\}$;

collects all parents of a given class c;

full descriptor of a class $c \in \mathrm{CLASS}$ is a structure $FD_c = (\mathrm{ATT}^*_c, \mathrm{OP}^*_c, \mathrm{navends}^*(c))$,

i. containing all attributes:

$$\mathrm{ATT}^*_c = \mathrm{ATT}_c \cup \bigcup_{c' \in \mathrm{parents}(c)} \mathrm{ATT}_{c'},$$

ii. user-defined operations:

$$\mathrm{OP}^*_c = \mathrm{OP}_c \cup \bigcup_{c' \in \mathrm{parents}(c)} \mathrm{OP}_{c'},$$

iii. and navigable role names:

$$\mathrm{navends}^*(c) = \mathrm{navends}(c) \cup \bigcup_{c' \in \mathrm{parents}(c)} \mathrm{navends}(c'),$$

defined for the class c and all its parents;

properties of the full descriptor:

i. Attributes are defined in exactly one class:

$\forall (a: t_c \rightarrow t, a': t_{c'} \rightarrow t' \in \mathrm{ATT}^*_c): (a=a' \Rightarrow t=t_{c'} \wedge t=t')$,

ii. An operation may only be defined once:

$\forall (\omega: t_c \times t_1 \times \ldots \times t_n \rightarrow t, \omega: t_{c'} \times t_1 \times \ldots \times t_n \rightarrow t' \in \mathrm{OP}^*_c): (t_c=t_{c'})$,

iii. Role names are defined in exactly one class:

$\forall\ c_1, c_2 \in \mathrm{parents}(c) \cup \{c\}: (c_1 \neq c_2 \Rightarrow \mathrm{navends}(c_1) \cap \mathrm{navends}(c_2) = \emptyset)$,

iv. Role names and attribute names must not conflict:

$\forall (a: t_c \rightarrow t \in \mathrm{ATT}^*_c) \wedge \forall r \in \mathrm{navends}^*(c): (a \neq r)$;

syntax of object models is a structure of the above defined elements:

$\mathcal{M} = (\mathrm{CLASS}, \mathrm{ATT}_c, \mathrm{OP}_c, \mathrm{ASSOC}, \mathrm{associates}, \mathrm{roles}, \mathrm{multiplicities}, \prec)$;

object identifiers (usually single letters combined with increasing indexes):

i. The set of object identifiers of a class $c \in \mathrm{CLASS}$ is defined by an infinite set

$\mathrm{oid}(c) = \{\underline{c}_1, \underline{c}_2, \ldots.\}$,

ii. The domain of a class $c \in \mathrm{CLASS}$ is defined as

$I_{\mathrm{CLASS}}(c) = \bigcup \{\mathrm{oid}(c') \mid c' \in \mathrm{CLASS} \wedge (c' \prec c \vee c' = c)\}$;

generalization hierarchy and **object identifiers** relationship:

$\forall c_1, c_2 \in \mathrm{CLASS}: c_1 \prec c_2 \Rightarrow I(c_1) \subseteq I(c_2)$;

links – each association $as \in \mathrm{ASSOC}$ with $\mathrm{associates}(as) = \langle c_1, \ldots, c_n \rangle$ is interpreted as the Cartesian product of the set of object identifiers of the participating classes:

$I_{Assoc}(as) = I_{CLASS}(c_1) \times ... \times I_{CLASS}(c_n)$,

where a link denoting a connection between objects is an element $l_{as} \in I_{Assoc}(as)$;

system state – (objects, links, and attribute values together constitute the state of a system at a discrete point of time); a system state of a model $\mathcal{M}$ is a structure:

$\sigma(\mathcal{M}) = (\sigma_{CLASS}, \sigma_{ATT}, \sigma_{ASSOC})$, where:

i. The finite sets $\sigma_{CLASS}(c)$ include all objects of a class $c \in CLASS$ existing in the system state: $\sigma_{CLASS}(c) \subset oid(c)$,

ii. Functions σ_{ATT} assign attribute values to each object:

$\sigma_{ATT}(a): \sigma_{CLASS}(c) \to I(t)$ for each $a: t_c \to t \in ATT^*_c$,

iii. The finite sets σAssoc contain links (satisfying multiplicities) connecting objects:

$\forall\, as \in ASSOC, \sigma_{Assoc}(as) \subset I_{Assoc}(as)$;

OCL types and operations – there are two groups of OCL types distinguished (see details in [83]):

i. Non-collection types:

- Basic types: $T_B = \{Integer, Real, Boolean, String\}$; their operations Ω_B;
- Enumeration types (user-defined): T_E, being sets of enumeration literals; their operations: Ω_E;
- Object types T_C derived from the UML classes definitions; their operations Ω_C;
- Special types: $T_S = \{OclAny, OclState, OclVoid\}$, their operations Ω_S, where:
  - OclAny is a supertype of all other types except collection types,
  - OclState, similar to T_E, is used to refer to state names in a state machine,
  - OclVoid is the subtype of all others types, representing the undefined value $\perp$;

ii. Collection types (grouped by the Collection(t) supertype): Set(t), Sequence(t), Bag(t) to describe collections of the value of a given type t and tuple type Tuple(l_1:t_1, ..., l_n:t_n) to describe the combination of values of different types t_1, ...t_n; their syntax and semantics were defined recursively; examples: a set (no duplicating elements) {2,3,6,1} is expressed in the OCL as Set{2,3,6,1}, a list <2,4,6> (ordered) as Sequence{2,4,6}, and a bag {{2,3,3,3,4,4,6}} as Bag{2,3,3,3,4,4,6}; a bag (i.e. a multi-set) may contain multiple copies of an element.

syntax and semantics – the approach and selected examples

i. The syntax of types and operations is represented by the data signature $\sum=(T, \Omega)$;

ii. The semantics of types in T is defined by mapping (interpretation I) that assigns a domain to each type,

ii. The semantics of operations in Ω is defined by mapping (interpretation I) that assigns a function to each operation;

Example of definition: **Syntax of basic types** is:

The set of basic types is $T_B = \{Integer, Real, Boolean, String\}$;

Example of definition: **Semantics of basic types** is:

$I(Integer) = Z \cup \{\perp\}$,

$I(Real) = R \cup \{\perp\}$,

$I(Boolean) = \{true, false, \perp\}$,

$I(String) = A^* \cup \{\perp\}$;

Example of definition: **Syntax of operations on the basic types**

The syntax of an operation is defined by a signature $\omega: t_1 \times ... \times t_n \rightarrow t$, which contains the operation symbol ω, a list of parameter types: $t_1, ..., t_n \in T$, and a result type $t \in T$;.

Example of definition: **Semantics of operations on the basic types**

The semantics of an operation with signature $\omega: t_1 \times ... \times t_n \rightarrow t$ is a total function

$I(\omega: t_1 \times ... \times t_n \rightarrow t): I(t_1) \times ... \times I(t_n) \rightarrow I(t)$;

e.g. (interpretation of the operation ω ='+' for adding two integer numbers i_1, i_2 is:

$I(+)(i_1, i_2) = i_1 + i_2$, (if $i_1 \neq \perp$ and $i_2 \neq \perp$), or $\perp$ otherwise;

Example of definition: **Syntax of enumeration types**

An enumeration type $t \in T_E$ is associated with a finite, non-empty set of enumeration literals by a function: $literals(t) = \{e_{1,t}, ..., e_{n,t}\}$;

Example of definition: **Semantics of enumeration types**

The semantics of an enumeration type $t \in T_E$ is a function $I(t) = literals(t) \cup \{\perp\}$;

Example of definition: **Semantics of an OCL predefined operation** $allInstances_t: \rightarrow Set(t)$ concerning enumeration type:

$\forall\ t \in T_E: I(allInstances_t()) = literals(t)$;

the same operation may concern the objects of a type $t \in T_C$, and is called the predefined operation for objects;

Example of definition: **Syntax of object types**

Let M be a model with a set CLASS of class names. The set T_C of object types is defined in such a way that $\forall\ c \in CLASS\ \exists\ t \in T_C$ has the same name as the class c; there are two functions defined for mapping a class to its type and vice versa:

$typeOf: CLASS \rightarrow T_C$,

$classOf: T_C \rightarrow CLASS$;

Example of definition: **Semantics of object types**

The semantics of object type $t \in T_C$ with $classOf(t) = c$ is defined as

$I(t) = I_{CLASS}(c) \cup \{\perp\}$;

Example of definition: **Semantics of attribute operations**

An attribute signature $a: t_c \rightarrow t$ in Ω_C is interpreted by an attribute value function $I_{Att}(a: t_c \rightarrow t): I(t_c) \rightarrow I(t)$ mapping objects of class c to a value of type t:

$I_{Att}(a: t_c \rightarrow t)(\underline{c}) = \sigma_{Att}(a)(\underline{c})$ (if $\underline{c} \in \sigma_{CLASS}(c)$), or $\perp$ otherwise;

Example of definition: **Syntax of navigation operations along association**

Let $\mathcal{M}$ be an object model syntax:

$\mathcal{M} = (CLASS, ATT_c, OP_c, ASSOC, associates, roles, multiplicities, \prec)$.

The set $\Omega_{nav}(c) \in \Omega_C$ of navigation operations for a class $c \in CLASS$ is defined in such a way that for each association $as \in participating(c)$ with $associates(as) = \langle c_1, \ldots, c_n \rangle$, $roles(as) = \langle r_1, \ldots, r_n \rangle$, and $multiplicities(as) = <\mathcal{M}_1, \ldots, \mathcal{M}_n>$ the following signatures are in $\Omega_{nav}(c)$:

$\forall i, j \in \{1, \ldots, n\}$ with $i \neq j$,

$c_i = c$, $classOf(t) = c$, $t_{ci} = typeOf(t_{ci})$, and $t_{cj} = typeOf(t_{cj})$:

i. if n=2 and $\mathcal{M}_j - \{0,1\} = \varnothing$ then $r_{j(as,ri)}: t_{ci} \rightarrow t_{cj} \in \Omega_{nav}(c)$; (the result type of the navigation over binary associations is the type of the target class if the multiplicity of target is 0..1 or 1);

ii. if n>2 or $\mathcal{M}_j - \{0,1\} \neq \varnothing$ then $r_{j(as,ri)}: t_{ci} \rightarrow Set(t_{cj}) \in \Omega_{nav}(c)$; (all non-binary associations and binary associations with all multiplicities other than those mentioned above induce the object of the source class links with multiple objects of the target class);

Example of definition: **Semantics of navigation operations along association**

Assuming that the set of objects of class c_j linked to an object $\underline{c}_j$ via association as is defined as:

$L(as)(\underline{c}_i) = \{ \underline{c}_i \mid (\underline{c}_1, \ldots, \underline{c}_i, \ldots, \underline{c}_j, \ldots, \underline{c}_n) \in \sigma_{ASSOC}(as)\}$;

the semantics of operations in $\Omega_{nav}(c)$ is defined as follows:

i. $I(r_{j(as,ri)}: t_{ci} \rightarrow t_{cj})(\underline{c}_i) = \underline{c}_j$ (if $\underline{c}_j \in L(as)(\underline{c}_i)$), or $\perp$ otherwise,

ii. $I(r_{j(as,ri)}: t_{ci} \rightarrow Set(t_{cj}))(\underline{c}_i) = L(as)(\underline{c}_i)$;

APPENDIX C. BASIC TERMS AND DEFINITIONS CONCERNING THE UMLSEC APPROACH TO MODELLING CRYPTOGRAPHY

This appendix includes the selected definitions and terms used in the IT product or system modelling, especially their cryptographic elements, according to the UMLsec [72]. Such products or systems are often based on cryptographic protocols. For that reason, this methodology adopts commonly used cryptographic terms and definitions that are also preferred in this monograph. A very exhaustive discussion concerning cryptography is contained in [80].

The UMLsec uses the algebra of cryptographic expressions Exp which is generated by the set $\mathsf{Keys} \cup \mathsf{Var} \cup \mathsf{Data}$, where:

- Keys – cryptographic keys (symmetric, when $K^{-1}=K$, asymmetric otherwise; $K \in \mathsf{Keys}$);
- Var – variables;
- Data – data values, including $\mathsf{Secrets}$; cryptographic protocols often use random values that ought to be used only once, called a **nonce**, which also belong to the set of secrets.

This algebra is based on the following operations, where "_" denotes a place for the argument:

$\_::\_$ – concatenation	$\mathsf{head}(\_)$ and $\mathsf{tail}(\_)$ – head and tail of concatenation
$\{\_\}_\_$ – encryption	$\mathsf{Dec}_\_(\_)$ – decryption
$\mathsf{Sign}_\_(\_)$ – signing	$\mathsf{Ext}_\_(\_)$ – extracting from signature
$\mathsf{Hash}(\_)$ – hashing	

Please note the following properties:

i. $\mathsf{Dec}_{K^{-1}}(\{E\}_K)=E;\ \forall E \in \mathsf{Exp} \wedge \forall K \in \mathsf{Keys};$

ii. $\mathsf{Ext}_K(\mathsf{Sign}_{K^{-1}}(E))=E;\ \forall E \in \mathsf{Exp} \wedge \forall K \in \mathsf{Keys};$

iii. $(E_1::E_2)::E_3=E_1::(E_2::E_3);\ \forall E_1, E_2, E_3 \in \mathsf{Exp};$

iv. $\mathsf{head}(E_1::E_2)=E_1$, $(\forall E_1, E_2 \in \mathsf{Exp})$ and $\mathsf{tail}(E_1::E_2)=E_2$, $(\forall E_1, E_2 \in \mathsf{Exp}$ such that there exist no such E, E' with $E_1= E::E')$; for other cases $\mathsf{head}()=\perp$ and $\mathsf{tail}()=\perp$.

For extracting the first, second, third, etc., element from the sequence, the following operations can be used:

i. **fst**(E) = head(E),

ii. **snd**(E) = head(tail(E)),

iii. **thd**(E) = head(tail(tail(E))), etc.

APPENDIX D. BASIC PRINCIPLES OF NAMING THE TERMS

Table D.1

Basic principles of naming the terms

Monograph text	**OCL code**	**Meaning**
`Generic`	`Generic`	An abstract UML class
`GenItem`	`GenItem`	A non abstract UML class
attribute	`attribute`	
operation()	`operation()`	
isCovered, parameter		The role names derived from class names *`IsCovered`* and *`Parameter`*
OACC.DAC_D0I0		The generic name issued by the *develsname()* operation
Roles, as, GenParAssoc, …		A calligraphic font is used for mathematical symbols (formal basis of the OCL)
$S_{Generic}$, $Set(t_{Generic})$, where $t_{Generic}$ is a type derived from class *`Generic`*	`Set(`*`Generic`*`)`	The set of classes of the given kind
"4" within names, e.g. *`Ogr4Pgr`*		An abbreviated word "for"

APPENDIX E. ELEMENTS OF THE SECURITY TARGET FOR A FIREWALL SYSTEM

The Appendix E exemplifies selected aspects of the presented there methodology, based on the IT Security Development Framework (ITSDF). The example deals with a simple firewall, called the FW system, developed on the basis of guidelines placed in "Annex D Worked Example: Firewall PP and ST" [60]. The presented example shows how these rough project ideas can be expressed with the use of facilities built in the ITSDF framework and its software implementation. As it was mentioned earlier, the elaboration process of the IT product or systems consists of:

- the IT security development process (expressed there by the ITSDF framework), leading to the creation of the Security Target (ST),
- TOE development process, where the given IT product or system is created, documented and prepared for the evaluation on the ST basis, with the rigor implied by the EAL level and with the use of the given technology.

The ITSDF framework contains two main groups of models:

- models of IT security development processes (`SM_SecurityModel`), expressing the elaboration of the security specifications, like ST, PP,
- models of the specification means (`SL_SecurityLibrary`), used to create these specifications.

Generally, the ITSDF uses three levels of description: informal (textual) level, semiformal (UML or developer's style) level, formal (OCL or mathematics-based) level. Moreover, the elements of the FW system will be exemplified with the use of the ITSDF-tool software tool which is an implementation of the ITSDF models. All above kinds of specification will be exemplified by the FW system example, though the developer's style will be used as the main specification method and will embrace all elements. Some of the description styles (levels) have practical meaning and can be met in the ST/PP specifications, and some have theoretical meaning and are used for internal model representation.

The following sections present activities and their results leading to the elaboration of the FW security target. The example concerns CC v.2.x which is compliant with the [60] standard. Some differences existing between the above version and the current one, i.e. CC v.3.1., can be ignored in this case.

The IT security development process is defined by the state machine (Fig. 2.6) and the related data models presented in the Fig. 2.4, refined in the chapters 4-10. They express the ST/PP general structure and all activities leading to filling this structure with the right data on the specific security project, like the FW system. Some of these data have informal and some semiformal character, like the predefined generics, functional and assurances components, taken from the specification means library – discussed in the chapter 3.

E.1. Elaboration of the ST introduction

Prior to the ST introduction model (Fig. 4.4), the ITSDF framework recommends the elaboration of the BCL model of the firewall system (Fig. 4.3). Going step by step, different project identifiers are assigned and different features are precisely, though informally, expressed. This stage, rather simple in comparison with others, is supported by the set of the electronic forms and simple project database implemented in the ITSDF tool.

Example E.1: ST instruction contents – informal description.

Apart from the FW and its ST identifiers and a short presentation of the functionality and connectivity, some additional information is needed on: security functionality (the FW is “security” product) and the FW environment, i.e. underlying hardware/software platform, the need for physical protection, different roles of the administrator and users, protected assets which are outside the FW, etc. □

E.2. Security environment (“Security problem definition” in the latest CC version)

The security environment data model is shown in the Fig. 5.2 and 5.3, while the elaboration process in the Fig. 5.4 and its refinements. To fulfil the data model, the generics expressed assets, subjects, threats, OSPs and assumptions are used (see Chapter 3).

Example E.2: Security environment specification – selected issues expressed by generics.

The firewall intermediates between the protected private network and the hostile public ones. The protected assets, which are data and services, are outside the FW, i.e. they are in the protected network. To simplify this issue, all assets are expressed with the use of only one generic, which can be used as the `paramDA` value for other generics, e.g. threats:

> ***DAE.ProtNet***. *Hosts, workstations, its data and services on the private network protected by the firewall.*

Subjects represent active entities, used as `paramS` values. For the considered firewall **two authorized** individuals are identified, representing administrators and users:

> ***SAU.FullAccAdmin***. *TOE administrator, having full access rights.*

SAU.NetUser.D1. *Distinguished user of the protected network or external, potentially hostile network.*

and **one unauthorized**, representing threat agents (please not an example of the generic derivation):

SNA.HighPotenIntrud.D1. *Attacker having high level skills, enough resources and deep motivation to perform a deliberate attack.*

To avoid bypassing the FW system, two assumptions of connectivity aspects were assigned:

AC.DualHomed. *The firewall has separate network adapters for every network connection.*

AC.FirewallConn.D1. *The firewall is assumed to be configured as the only network connection between the private network and the hostile network.*

Moreover, three assumptions (note an example of the parameterization of generics) of personal aspects should be satisfied:

AP.OnlyAdminAccess. *Only administrators [paramS <= SAU.FullAccAdmin] can access the firewall.*

AP.TrustAdmin. *It is assumed that one or more authorized administrators [paramS <= SAU.FullAccAdmin] are assigned who are competent to manage the TOE and the security of the information it contains, and who can be trusted not to deliberately abuse their privileges so as to undermine security.*

AP.NoDistUsers. *Users [paramS <= SAU.NetUser.D1] are considered in the same way as the sources of security breaches.*

The most important issue is the threats specification (simplified there). Three kinds of direct attacks are considered:

TDA.IllegAcc. *An attacker [paramS <= SNA.HighPotenIntrud.D1] on the hostile network may exploit flaws in service implementations (e.g. using a 'well known' port number for a protocol other than the one defined to use that port) to gain access to hosts or services [paramDA <= DAE.ProtNet].*

TDA.FwlAdminImpers. *An attacker [paramS1 <= SNA.HighPotenIntrud.D1] may gain access to the firewall by impersonating an administrator [paramS2 <= SAU.FullAccAdmin].*

TDA.NewAttMeth. *Attackers [paramS <= SNA.HighPotenIntrud.D1] on the hostile network exploiting new, previously unknown attack methods, e.g. using previously trustworthy services.*

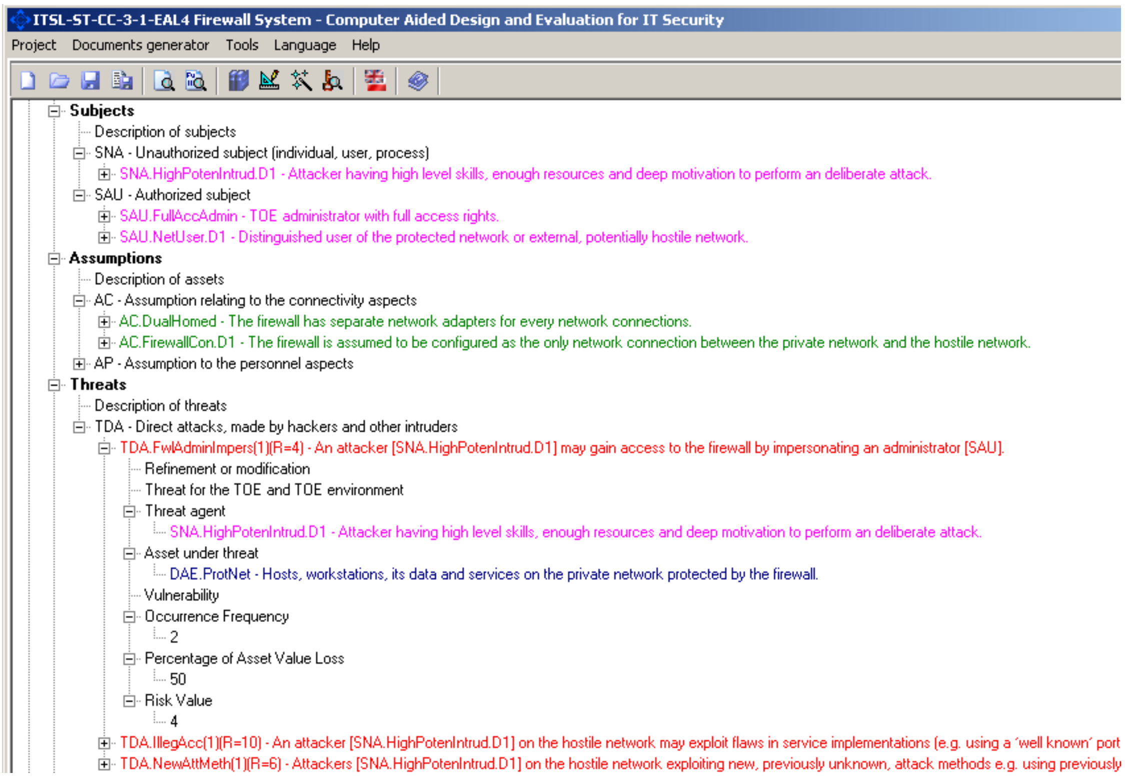
Please note that *TDA.FwlAdminImpers* has 2 `paramS` parameters (not serviced by the ITSDF-tool yet). For the FW system the security problem is expressed by threats, and no OSPs generics were used. □

Example E.3: Security environment – generics from the Example E.2 within the ITSDF-tool.

The Fig. E.1 presents the security environment specification in the ITSDF-tool, as "the design tree". Please note the above mentioned (Example E.2) subjects, assumptions and threats, including a few derived ones, and three threat generics, among which one is expanded. It contains refinement, the attribute saying that the threat concerns both the TOE and its environment, two assigned parameters, exploited vulnerability and the risk attributes. The asset generic with the *assetValue* attribute is assigned as the parameter of the threat generic. With respect to the threat character and the threat agent (expressed by the `paramS`)

the *eventLikelihood*[17] and *assetValLoss*[18] abilities are assessed by the developer, using predefined scales for the given project.
See the Definitions 3.11 and 3.17 introducing these parameters and the example of the OCL definition of the `riskValueAssess()` operation placed in the Section 5.6.
Note that:

$$riskValueAsses = assetValue * eventLikelihood * assetValLoss. \qquad (1)$$


ITSL-ST-CC-3-1-EAL4 Firewall System - Computer Aided Design and Evaluation for IT Security
Project
Documents generator
Tools
Language
Help

```
Subjects
    Description of subjects
    SNA - Unauthorized subject (individual, user, process)
        SNA.HighPotenIntrud.D1 - Attacker having high level skills, enough resources and deep motivation to perform an deliberate attack.
    SAU - Authorized subject
        SAU.FullAccAdmin - TOE administrator with full access rights.
        SAU.NetUser.D1 - Distinguished user of the protected network or external, potentially hostile network.
Assumptions
    Description of assets
    AC - Assumption relating to the connectivity aspects
        AC.DualHomed - The firewall has separate network adapters for every network connections.
        AC.FirewallCon.D1 - The firewall is assumed to be configured as the only network connection between the private network and the hostile network.
    AP - Assumption to the personnel aspects
Threats
    Description of threats
    TDA - Direct attacks, made by hackers and other intruders
        TDA.FwlAdminImpers(1)(R=4) - An attacker [SNA.HighPotenIntrud.D1] may gain access to the firewall by impersonating an administrator [SAU].
            Refinement or modification
            Threat for the TOE and TOE environment
            Threat agent
                SNA.HighPotenIntrud.D1 - Attacker having high level skills, enough resources and deep motivation to perform an deliberate attack.
            Asset under threat
                DAE.ProtNet - Hosts, workstations, its data and services on the private network protected by the firewall.
            Vulnerability
            Occurrence Frequency
                2
            Percentage of Asset Value Loss
                50
            Risk Value
                4
        TDA.IllegAcc(1)(R=10) - An attacker [SNA.HighPotenIntrud.D1] on the hostile network may exploit flaws in service implementations (e.g. using a 'well known' port
        TDA.NewAttMeth(1)(R=6) - Attackers [SNA.HighPotenIntrud.D1] on the hostile network exploiting new, previously unknown, attack methods e.g. using previously
```

Fig. E.1. The ITSDF-tool – the part of the design three dealing with the FW firewall security environment
Rys. E.1. Narzędzie ITSDF – fragment drzewka projektu systemu zaporowego FW

[17] "Occurrence Frequency" in the ITSDF-tool.
[18] "Percentage of the Asset Value Loss" in the ITSDF-tool.

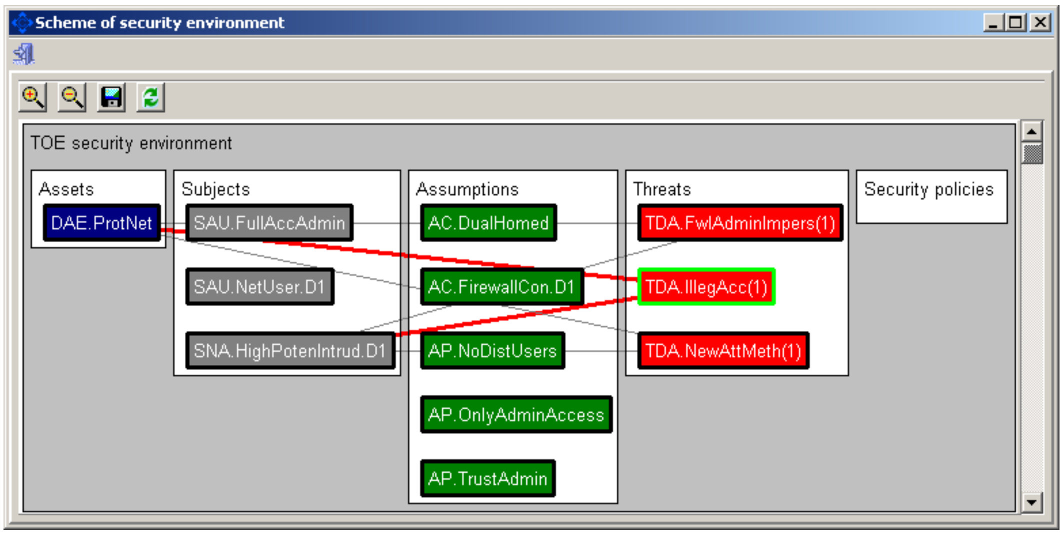


Fig. E.2. *Visualization of the security environment in the software tool based on the ITSDF framework*

Rys. E.2. *Wizualizacja otoczenia zabezpieczeń w narzędziu będącym implementacją systemu ITSDF*

Due to the substitution of parameters values concerning generics: *DAE.ProtNet* and *TDA.FwlAdminImpers* respectively, the obtained *riskValueAssess*[19] is 4 (in the predefined scale):

$$riskValueAssess = 4*2*0.5 = 4. \tag{2}$$

In this example no generic refinements were made. Please note that for any threats the developer can add more info on the exploited vulnerabilities. The presented generics are introduced by the developer from the generics/components library, where hundreds of predefined generics and all Common Criteria components are placed.

The Fig. E.2 shows how the elements of the security environment specification are visualized by the ITSDF-tool.

Please note generics and relations concerning the assigned parameters. For the highlighted *TDA.IllegAcc*[20] two relations exist:

(*TDA.IllegAcc*, *SNA.HighPotenIntrud.D1*) and (*TDA.IllegAcc*, *DAE.ProtNet*). □

E.3. Security objectives

The main security objectives data model is shown in the Fig. 6.3. The elaboration process is presented in the Fig. 6.8 and its refinements. To fulfil the data model, different kinds of security objectives generics are used (Fig. 3.8, Definition 3.27). The security objectives provide the solution to the security problem definition. The security objectives elaboration has two steps. In the first step all security objectives are placed in the common set, then the security objectives belonging to this set are analyzed and moved to one of two other sets embracing the TOE security objectives or the TOE environment security objectives. This

[19] “Risk Value” in the ITSDF-tool.

[20] In the ITSDF-tool the instance number is placed in brackets.

process, called "trade-off", supported by the ITSDF-tool, partitions the "security responsibility" between the TOE and its environment and decides about the TOE shape, degree of automation, maintenance cost, evaluation cost, etc. Please note that only the TOE security requirements will be transformed to functional requirements, the latter to the security functions (TSFs), evaluated against the declared EAL and implemented within the IT product or system. As for the firewall, more TOE security objectives mean better automation and easier administration but higher cost of evaluation influencing the overall product cost. On the other hand, more security environment objectives declared mean cheaper products which, however, create more difficult conditions for the exploitation environment and administration.

Example E.4: Security objectives specification – selected issues expressed by generics.

The main security objectives for the FW system, related to the ***TDA.IllegAcc*** threat, concern the access restrictions and control between the protected private network and the hostile public ones. Because this access control can be performed on different levels, there are **three main** TOE security objectives assigned to cover this issue:

OACC.LmtIPAddr. *The firewall enforces access control by limiting the valid range of addresses expected on each of the private and hostile networks (i.e. an external host cannot spoof an internal host).*

OACC.LmtPortHost. *The firewall enforces access control by limiting the hosts and service ports that can be accessed from, respectively, the hostile and private networks.*

OACC.OnProxyAuth. *The firewall must, for certain specified services [paramDA <= *[21]] of proxy application, be capable of requiring authentication of the end user [paramS <= SAU.NetUser.D1] prior to establishing a through connection.*

and **three others**, concerning the FW audit and management facilities, and data sanitizing:

OADT.RecSecEvents. *The TOE will provide the means of recording any security relevant events, so as to assist an administrator in the detection of potential attacks or misconfiguration of the TOE security features that would leave the TOE susceptible to attack, as well as to hold users accountable for any security-relevant actions.*

OSMN.SecManAdmin. *The TOE will provide facilities to enable an authorized administrator [paramS <= SAU.FullAccAdmin] to effectively manage the TOE and its security functions, and will ensure that only authorized administrators are able to access such functionality.*

ODEX.SanitData. *Sanitizing data objects containing hidden or unused data. Sanitize data objects that may contain hidden data when they are exported from the TOE in order to inhibit steganographic smuggling.*

The above mentioned TOE security objectives should be **supported by** procedural or technical solutions expressed by the applied security objectives dealing with the environment (the first has procedural, the second technical character):

OSMN.SecConfManag. *Security-relevant configuration management. Managing and updating system security policy data and enforcement functions, and other security-relevant configuration data, in accordance with organizational security policies.*

[21] „*" (the value not assigned); for the FW means: „Any proxy service".

OADT.AuditManage. *Administrators of the TOE [paramS <= SAU.FullAccAdmin] must ensure that audit facilities are used and managed effectively. In particular: action must be taken to ensure continued audit logging, e.g. by regular archiving of logs before audit trail exhaustion to ensure sufficient free space. Audit logs should be inspected on a regular basis, and appropriate action should be taken to detect breaches of security or events that are likely to lead to a breach in the future.*

The second problem for the FW system concerns the administrator being impersonated by the intruder – the problem expressed by the ***TDA.FwlAdminImpers*** threat. By means of the above mentioned audit and management facilities, the following solution is proposed:

OACC.AdminAuth. *Authentication of the firewall administrator [paramS <= SAU.FullAccAdmin].*

The third FW security problem (***TDA.NewAttMeth***), concerning continuously occurring new methods of attacks, discovered vulnerabilities, etc. can be solved by organizational means:

OSMN.AuthNotif. *Appropriate authorities shall be immediately notified of any new threats or vulnerabilities impacting systems that process their data.*

OSMN.Awareness. *The organization implements users' awareness programme.* □

Example E.5: The security problem of the FW system and its solution – a kind of specification which is alternative to the Example E.4 and uses the UML object diagram.

All generics presented in the Examples E.2 and E.4 have their internal representations as the security UML objects, i.e. different kinds of `GenItem`: `ACItem`, `TDAItem`, `OACCItem`, etc. Please note two name conventions: the developer's style (Definition 3.2, Definition 3.35) and the UML style `object name:Object type` (Definition 3.35 – the operation `develsname()`). For example, the *SNA.HighPotenIntrud.D1* generic (developer's style) can be expressed as a UML object `HighPotenIntrud_D1:SNAItem`. Please note (Fig. 3.13) that when a generic (or component) is taken from the library to the security specification, it changes its `stateAttribute` from `DEFINED` to `ASSIGNED`. The UML object diagram (Fig. E.3) can be considered as a picture of the general class diagrams expressing a concrete situation dealing with the FW system security environment and objectives.

For example, the parameterization association classes `ParamDA4T`, `ParamS4T` (Fig.3.15) have some instances represented by the link `parameter`, and covering association classes (such as `Ogr4Tgr` shown in the Fig. 3.18) have some instances represented by the link `covers`. Justification is required for the security associations concerning mapping security issues of the neighbouring IT security development stages, i.e. the "covering associations". Please note the attributes of the association classes, e.g. in the Fig. 3.18. Justification is needed for any covering decision made by the developer. For example, the justification of any security objectives covering the given threat is provided as the last action in the Fig. 6.9.

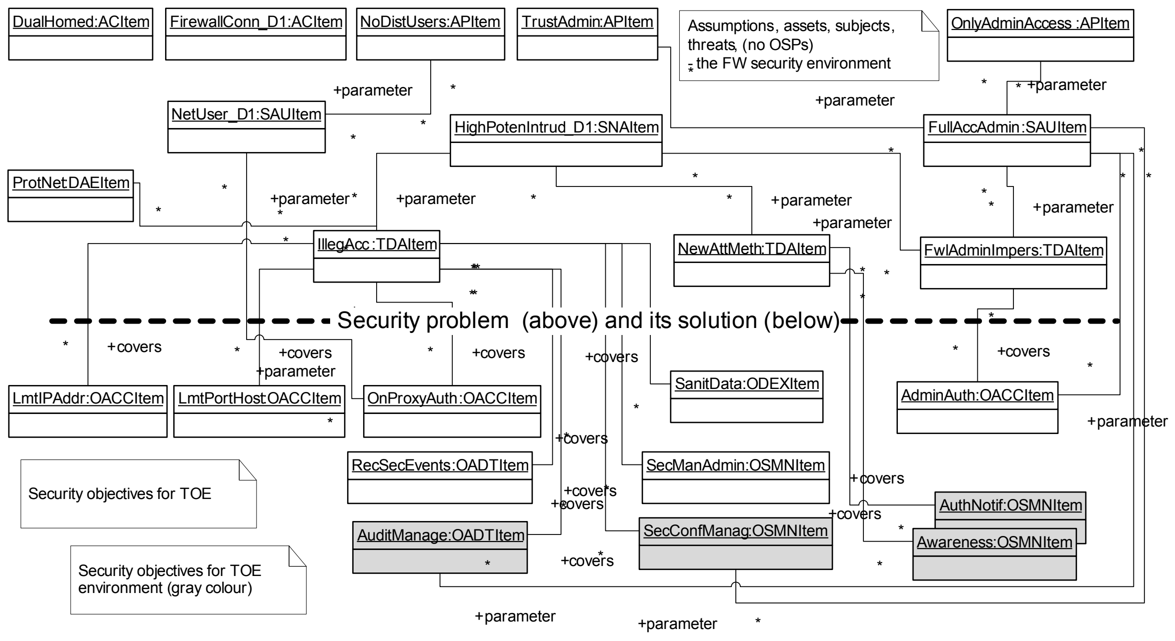


Fig. E.3. Security problem (environment) and its solution (objectives) for the FW system on the UML object diagram

Rys. E.3. Sformułowany dla systemu FW problem bezpieczeństwa (otoczenie zabezpieczeń) oraz jego rozwiązanie w postaci celów zabezpieczeń, pokazane jako diagram obiektów UML

The justifications of any item makes the security objectives rationale process (Section 10.1) easier, providing an optimized set of security objectives necessary and sufficient to cover the security problem expressed by the security environment. The selection of the security objectives is supported by simple risk analyzing facilities discussed in the Example E.3. Please note risk values displayed for any threat in the Fig. E.1, e.g.: *TDA.IllegAcc(R=10)*. The risk rank is the basis to map a security objective of given properties, to support it by other objectives, or to refine it. □

Example E.6: The security problem of the FW system and its solution – a kind of specification which is alternative to the Example E.4 and E.5 and uses the model mathematical representation.

The example shows the mathematical representation of the security problem definition and security objectives specifications, as parts of the FW security target (Definitions: 5.1, 6.1). The security environment specification includes all "assigned" Sgr, DAgr, Tgr, Pgr, Agr generics (Definition 3.9), i.e. $S_{TOESecEnvItems}$. It also includes the set of pairs expressing parameterization relation, representing Sgr or DAgr generics assigned to the specified Tgr, Pgr, Agr generics, i.e. the $S_{TOESecEnvPars}$. For the considered firewall system the security environment specification contains the items listed below. In this case the generics are represented as UML objects of the given type. Please compare this notation with the alternative one, i.e. the developer's style notation used in the Example 10.4.

$S_{TOESecEnv} = <S_{TOESecEnvItems}, S_{TOESecEnvPars}>$

$S_{TOESecEnvItems} = S_{Subjects} \cup S_{Assets} \cup S_{Threats} \cup S_{OSPs} \cup S_{Assumpt}$, where:

$S_{Subjects}$ = Set{`FullAccAdmin:SAUItem`, `NetUser_D1:SAUItem`, `HighPotenIntrud_D1:SNAItem`},

S_{Assets} = Set{`ProtNet:DAEItem`},

$S_{Threats}$ = Set{IllegAcc:TDAItem, FwlAdminImpers:TDAItem, NewAttMeth:TDAItem},

S_{OSPs} = ∅,

$S_{Assumpt}$ = Set{DualHomed:ACItem, FirewallConn D1:ACItem, OnlyAdminAccess:APItem, TrustAdmin:APItem, NoDistUsers:APItem}.

$S_{TOESecEnvPars}$
= {ParamDA4T, ParamS4T, ParamDA4P, ParamS4P, ParamDA4A, ParamS4A},

ParamDA4T = Set{(ProtNet:DAEItem, IllegAcc:TDAItem)},

ParamS4T = Set{(HighPotenIntrud D1:SNAItem, IllegAcc:TDAItem),
(HighPotenIntrud D1:SNAItem, FwlAdminImpers:TDAItem),
(FullAccAdmin:SAUItem, FwlAdminImpers:TDAItem),
(HighPotenIntrud D1:SNAItem, NewAttMeth:TDAItem)},

ParamS4A = Set{(FullAccAdmin:SAUItem, OnlyAdminAccess:APItem),
(FullAccAdmin:SAUItem, TrustAdmin:APItem),
(NetUser D1:SAUItem, NoDistUsers:APItem)},

ParamDA4P = ParamS4P = ParamDA4A = ∅,

The security objectives specification includes three elements. The first represents all "assigned" Ogr generics (Definition 6.1), i.e. $S_{SecObjItems}$. The second is the set of pairs expressing parameterization relation, representing Sgr or DAgr generics assigned to the specified Ogr generics, i.e. the $S_{SecObjPars}$. The third element is the set of pairs expressing how the security environment elements are covered by the assigned security objectives, i.e. $S_{SecObjAssocs}$.

$S_{SecObj} = <S_{SecObjItems}, S_{SecObjPars}, S_{SecObjAssocs}>$.

The security objectives may concern the TOE ($S_{TOE\_ITObjectives}$) or the technical ($S_{EnvirITObjectives}$) or procedural ($S_{EnvirAuxObjectives}$) aspects of its environment:

$S_{SecObjItems} = S_{TOE\_ITObjectives} \cup S_{EnvirITObjectives} \cup S_{EnvirAuxObjectives}$.

For the FW system the following security objectives were specified:

$S_{TOE\_ITObjectives}$ = Set{LmtIPAddr:OACCItem, LmtPortHost:OACCItem, OnProxyAuth:OACCItem, RecSecEvents:OADTItem, SecManAdmin:OSMNItem, SanitData:ODEXItem, AdminAuth:OACCItem},

$S_{EnvirITObjectives}$ = Set{AuditManage:OADTItem},

$S_{EnvirAuxObjectives}$ = Set{SecConfManag:OSMNItem, AuthNotif:OSMNItem, Awareness:OSMNItem}.

The parameterization association contains the following pairs:

$S_{SecObjPars}$ = {ParamDA4O, ParamS4O}, where:

ParamDA4O = Set{(paramDA[22], OnProxyAuth:OACCItem)},

ParamS4O = Set{(NetUser D1:SAUItem, OnProxyAuth:OACCItem),
(FullAccAdmin:SAUItem, SecManAdmin:OSMNItem),
(FullAccAdmin:SAUItem, AuditManage:OADTItem),
(FullAccAdmin:SAUItem, AdminAuth:OACCItem)},

The association concerning mapping (covering) contains the following pairs:

$S_{SecObjAssocs}$ = {Ogr4Tgr, Ogr4Pgr, Ogr4Agr}, where:

Ogr4Tgr = Set{(LmtIPAddr:OACCItem, IllegAcc:TDAItem),
(LmtPortHost:OACCItem, IllegAcc:TDAItem),

[22] not assigned; for the FW means: „Any proxy service".

(OnProxyAuth:OACCItem, IllegAcc:TDAItem),
(RecSecEvents:OADTItem, IllegAcc:TDAItem),
(SecManAdmin:OSMNItem, IllegAcc:TDAItem),
(SanitData:ODEXItem, IllegAcc:TDAItem),
(SecConfManag:OSMNItem, IllegAcc:TDAItem),
(AuditManage:OADTItem, IllegAcc:TDAItem),
(AdminAuth:OACCItem, FwlAdminImpers:TDAItem),
(AuthNotif:OSMNItem, NewAttMeth:TDAItem),
(Awareness:OSMNItem, NewAttMeth:TDAItem)},

Ogr4Pgr = *Ogr4Agr* = $\varnothing$. □

The mathematical model is used as the internal data representation within the ITSDF framework.

E.4. Security requirements

The security requirements data model was shown in the Fig. 7.3, and the related specification workout in the Fig. 7.7 and its refinements. Please note different kinds of security requirements. The specification means for this stage are predefined in the Common Criteria catalogues. CC Part 2 contains functional components, while Part 3 assurance components and EALs definitions. For both kinds of CC components the UML/OCL models were created and implemented in the ITSDF-tool library, compatible with the generics models (Section 3.2). Security requirements for the environment can be expressed by CC components for a well defined IT environment or by the predefined RE-group of generics (Definitions: 3.29 and 3.30) in other case.

The selection of security assurance requirements (SARs) is based on the predefined assurance packages corresponding to particular EAL levels. The ITSDF framework allows to merge these predefined sets of requirements with the requirements from other sources, such as those deliberately selected by the developer or influenced by the specific security objectives. Merging the selected SAR and its depending SARs (defined by the CC) with the well composed assurance package is not easy and should be done carefully (Fig. 7.12). It leads to the EALn+ assurance level. For the considered firewall system the EAL4 was chosen. Please note that for an ST project the ASE class requirements are automatically attached (APE class for the PP).

The preparation of the SOF-claims for any functional requirement related to the permutational or probabilistic mechanism (for EAL>1) is supported by the ITSDF framework and its software implementation. In the latest CC version the SOF claims are withdrawn (replaced by the extended vulnerability analyses).

Example E.7: Security functional requirements (SFRs) for the FW system – selected issues expressed by the CC functional components or REgr generics.

For any main TOE security objective the right functional components (principal ones) are assigned, supplemented by dependent components and assigned intentionally by the developer. This set of requirements is supplemented by the requirements influenced by the supporting objectives – for the TOE and its environment. Please note that the ITSDF framework has predefined security associations ("proposing"), covering the given security objectives by the right components, called the supporting chains. The FW firewall will not provide the full security functionality and it will be based on the functionality provided by its IT environment, i.e. operating system (logging, audit). To sum up, the principal security requirements for the considered firewall example are selected as follows:

The TOE security objective ***OACC.LmtIPAddr*** is covered by:

> ***FDP_ACF.1*** *(Security attribute based access control),*
>
> ***FDP_ACC.2*** *(Complete access control),*

supported by the following component (assigned by the developer to avoid bypassability of this access control):

> ***FPT_RVM.1*** *(Non-bypassability of the TSP) ensures that these functions are always invoked when required.*

The TOE security objective ***OACC.LmtPortHost*** is covered by:

> ***FDP_IFF.1*** *(Simple Security Attributes),*
>
> ***FDP_IFC.2*** *(Complete Information Flow Control).*

The TOE security objective ***OACC.OnProxyAuth*** is covered by:

> ***FIA_UAU.2*** *(User Authentication Before Any Action),*
>
> ***FIA_UID.2*** *(User Identification Before Any Action).*

The TOE security objective ***OADT.RecSecEvents*** is covered by:

> ***FAU_GEN.1*** *(Audit Data Generation),*
>
> ***FAU_ARP.1*** *(Security Alarms),*

supported by the requirement addressed to the environment (implied by the objective ***OADT.AuditManage***):

> ***RENIT.AuditManage.*** *Administrators of the TOE [paramS <= SAU.FullAccAdmin] must ensure that audit facilities are used and managed effectively. In particular: action must be taken to ensure continued audit logging, e.g. by regular archiving of logs before audit trail exhaustion to ensure sufficient free space. Audit logs should be inspected on a regular basis, and appropriate action should be taken to detect breaches of security or events that are likely to lead to a breach in the future.*

The TOE security objective ***OSMN.SecManAdmin*** is covered by:

> ***FMT_SMR.1*** *(Security Management Roles),*
>
> ***FMT_MSA.1*** *(Management of Security Attributes),*
>
> supported by the requirement addressed to the environment (implied by the objective ***OSMN.SecConfManage***):
>
> ***RENIT.SecConfManage.*** *Security-relevant configuration management. Managing and updating system security policy data and enforcement functions as well as other security-relevant configuration data, in accordance with organizational security policies.*

The TOE security objective ***ODEX.SanitData*** is covered by:

> ***FDP_RIP.1*** *(Subset residual information protection).*

The TOE security objective ***OACC.AdminAuth*** is covered by (Note the iteration; these two components concern the authentication of the administrator, not the users as it was specified above for the objective ***OACC.OnProxyAuth***):

> ***FIA_UAU.2*** *(User Authentication Before Any Action),*
>
> ***FIA_UID.2*** *(User Identification Before Any Action),*

supported by the requirement ensuring non-bypassibility of authentication and generating alarms in such cases:

FPT_RVM.1 *(Non-bypassability of the TSP),*

FIA_AFL.1 *(Authentication Failure Handling).*

The TOE environment security objective ***OSMN.AuthNotif*** is covered by the requirement derived from the commonly used ISO/IEC 27001 standard:

RENIT.A12_6_1_ISO/IEC27001. *Control of technical vulnerabilities.*

The TOE environment security objective ***OSMN.Awareness*** is covered by the requirement derived from the commonly used ISO/IEC 27001 standard:

RENIT.UserAwarn. *User awareness and proper operation regulations.* □

This example is simplified and does not consider the security audit and security management issues, as well as component dependency analyses.

Example E.8: Security functional requirements (SFRs) for the FW system – the visualization of the components and generics from the Example E.7 within the ITSDF-tool (Fig. E.4).

The figure shows how the elements of the security functional requirements (the right side), elaborated on the security objectives basis (the left side), are visualized by the ITSDF-tool. The rectangles express the specification items (generics, components), while the lines express the associations (pairs) concerning mapping (Definition 7.1):

$S_{SecReqsAssocs} = \{\texttt{FunSec4Ogr}, \texttt{AssSec4Ogr}, \texttt{REgr4Ogr}\}.$

For example, thick lines going out of the highlighted *OADT.RecSecEvents* objective, point at the requirements *FAU_ARP.1* and *FAU_GEN.1* (implying 2 `FunSec4Ogr` pairs) and *RENIT.AuditManage* (implying 1 `REgr4Ogr` pair), all together covering this objective. For the FW system no SARs were derived directly from security objectives ($\texttt{AssSec4Ogr} = \varnothing$).

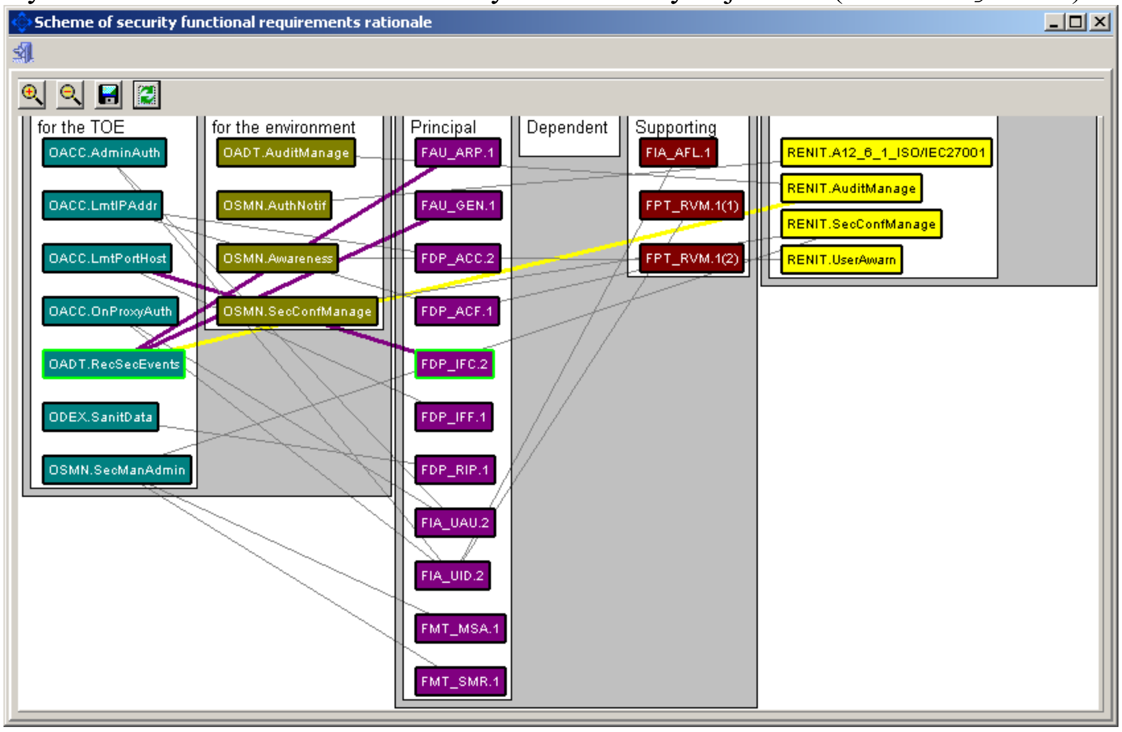


Fig. E.4. *Visualization of the security functional requirements in the software tool based on the ITSDF framework*

Rys. E.4. *Wizualizacja funkcjonalnych wymagań bezpieczeństwa w narzędziu będącym imple-*

mentacją systemu ITSDF

These SARs will be assigned on the declared EAL level basis which can be seen in the Example 9. Please note that by highlighting the *FDP_IFC.2* component the developer can see which objective is covered by this component, here: *OACC.LmtPortHost* (another example of the `FunSec4Ogr` pair). Please note that the FW example is simplified and for this reason no dependent components were assigned there. □

Example E.9: Security assurance requirements (SARs) for the FW system.

As it was mentioned earlier the selection of the assurance requirements for the TOE is very easy with a predefined EAL package. For the considered firewall the EAL4 was chosen, without augmentation or additions. It implies the set of SARs presented in the Table E.1. Please see the comments concerning the ADV class decomposition near the Fig. 3.11. Some problems may occur when the developer tries to add extra components or to replace some of them by more rigorous ones (EAL4+). In this case an extra analysis is required [40]. □

Table E.1

Assurance components of EAL4 declared for the FW system [40]

Assurance component	Description
ACM – Configuration management	
ACM_AUT.1	Partial CM automation
ACM_CAP.4	Generation support and acceptance procedures
ACM_SCP.2	Problem tracking CM coverage
ADO – Delivery and operation	
ADO_DEL.2	Detection of modification
ADO_IGS.1	Installation, generation, and start-up procedures
ADV – Development	
ADV_FSP.2	Fully defined external interfaces
ADV_HLD.2	Security enforcing high-level design
ADV_IMP.1	Subset of the implementation of the TSF
ADV_LLD.1	Descriptive low-level design
ADV_RCR.1	Informal correspondence demonstration
ADV_SPM.1	Informal TOE security policy model
AGD – Guidance documents	
AGD_ADM.1	Administrator guidance
AGD_USR.1	User guidance
ALC – Life cycle support	
ALC_DVS.1	Identification of security measures
ALC_LCD.1	Developer-defined life-cycle model
ALC_TAT.1	Well defined development tools
ATE – Tests	
ATE_COV.2	Analysis of coverage
ATE_DPT.1	Testing: high-level design
ATE_FUN.1	Functional testing
ATE_IND.2	Independent testing – sample
AVA –Vulnerability assessment	
AVA_MSU.2	Validation of analysis

AVA_SOF.1	Strength of TOE security function evaluation
AVA_VLA.2	Independent vulnerability analysis

E.5. Security functions of the TOE summary specification (TSS)

The data model of the TOE summary specification was shown in the Fig. 8.2, while the related specification workout was presented in the Fig. 8.3. To specify the security functions, to say more precisely – the Trusted Security Functions (TSFs), the F-group of generics was created (Definitions: 3.32 and 3.33). The functional security requirements are grouped (1 or more principal SFRs with the supporting ones) and to each group the security function is assigned representing a small set of closely related functionalities.

Each TSF can be considered as an elementary TOE module. These modules, created with the use of the assumed technology, can be implemented with the different rigour applied, depending on the EAL level. The level of rigour depends directly on the used ADV class components. The ITSDF system supports the justification of the SOF claims implementation within the security functions (for CC v. 2.x).

Example E.10: Security functions (TSFs) for the FW system expressed by generics.

The considered FW system includes 6 TSFs. They were defined around the main TOE security objectives (in this example security functions are labelled):

TSF1: F.LmtIPAddr. *Function responsible for IP address control between hostile and protected networks, using: apparent source IP address or host name and destination IP address or host name.*

This function is related to the TOE security objective ***OACC.LmtIPAddr*** and the following group of the SFRs:

FDP_ACF.1 *(Security attribute based access control),*

FDP_ACC.2 *(Complete access control),*

FPT_RVM.1 *(Non-bypassability of the TSP).*

TSF2: F.LmtPortHost. *Function responsible for port number control between hostile and protected networks, using apparent source port number and destination port number.*

This function is related to the TOE security objective ***OACC.LmtPortHost*** and the following group of the SFRs:

FDP_IFF.1 *(Simple Security Attributes),*

FDP_IFC.2 *(Complete Information Flow Control)*

FPT_RVM.1 *(Non-bypassability of the TSP).*

TSF3: F.OnProxyAuth. *Function responsible for authentication of the end user [paramS <= SAU.NetUser.D1] prior to establishing a through connection for specified services [paramDA<= *].*

This function is related to the TOE security objective ***OACC.OnProxyAuth*** and the following group of the SFRs:

FIA_UAU.2 *(User Authentication Before Any Action),*

FIA_UID.2 *(User Identification Before Any Action).*

FPT_RVM.1 *(Non-bypassability of the TSP),*

FIA_AFL.1 *(Authentication Failure Handling).*

TSF4: F.AdminAuth. *Function responsible for the firewall administrator access control, ensuring that only authorized [paramS <= SAU.FullAccAdmin] are able to access the firewall functionality. The detailed functionality: system login (identification, authentication), administrator accountability, logout.*

This function is related to the TOE security objective ***OACC.AdminAuth*** and the following group of the SFRs:

FIA_UAU.2 *(User Authentication Before Any Action),*

FIA_UID.2 *(User Identification Before Any Action).*

FPT_RVM.1 *(Non-bypassability of the TSP),*

FIA_AFL.1 *(Authentication Failure Handling).*

TSF5: F.AuditFacilities. *Function responsible for recording security related events and their management for audit purposes. These events may concern: IP addresses limitation, port number limitation, users' authentication on proxy for the selected network services, administrator's login, operations, and logout.*

This function is related to the TOE security objective ***OADT.RecSecEvents*** and the following group of the SFRs:

FAU_GEN.1 *(Audit Data Generation),*

FAU_ARP.1 *(Security Alarms).*

TSF6: F.FirewallManagement. *Function responsible for effective management of the TOE and its security functions. The firewall administrator [paramS <= SAU.FullAccAdmin], and only the firewall administrator, can perform the following functions: display and modify the firewall access control parameters, initialize and modify user authentication data, display and modify user attributes, select events to be audited, identify the subset of auditable events deemed to indicate a possible or imminent security violation, associate separate authentication mechanisms with specific authentication events, verify the integrity of the firewall.*

This function is related to the TOE security objectives ***OACC.SecManAdmin***, ***ODEX.SanitData*** and the following group of the SFRs:

FMT_SMR.1 *(Security Management Roles),*

FMT_MSA.1 *(Management of security attributes),*

FDP_RIP.1 *(Subset residual information protection).* □

Example E.11: Security functions (TSFs) for the FW system – the visualization of the components and generics from the Example E.10 within the ITSDF-tool.

The Fig. E.5 shows how the security functions (the right side), elaborated on the basis of security functional requirements (the left side), are visualized by the ITSDF-tool. The visualization is related to the mathematical model as well.

The rectangles express the specification items (Fgr generics, functional components), while the lines express the associations (pairs) concerning mapping (Definition 8.1):

$S_{TSS\_SecAssocs} = \{\mathtt{Fgr4FunSec}\}$.

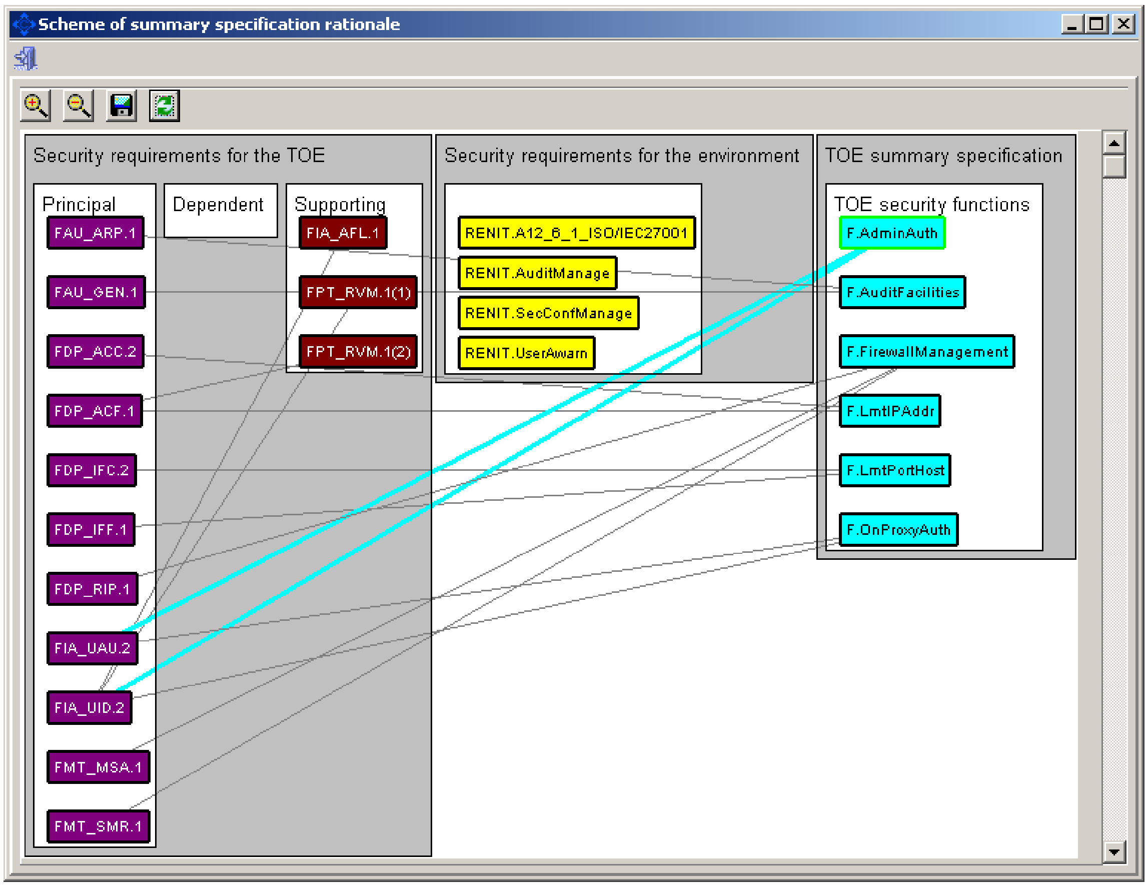


Fig. E.5. Visualization of the security functions in the software tool based on the ITSDF framework
Rys. E.5. Wizualizacja funkcji zabezpieczających za pomocą narzędzia ITSDF-tool

For example, thick lines going out of the highlighted *F.AdminAuth* function, point at the requirements *FIA_UID.2* and *FIA_UAU.2* (implying 2 `Fgr4FunSec` pairs), concerning the identification and authentication of the firewall administrator.

□

The ITSDF-tool allows to highlight any number of relations at any time. The visualization may concern all IT security development stages, e.g. from the given threat, through covering its objectives, assigned requirements, to the security function which is a countermeasure against this threat. The basic model for the visualization was shown in the Fig. 10.13, though the applied generics and components are specific for particular designs.

The Section 10.4 includes some examples of the security model operations allowing to investigate the model features. This is important for the model optimization during its iterative and recursive elaboration process, leading to the complete ST rationale. The same kind of operations can be done on the FW security model. Let us see some examples.

Example E.12: Operations on the firewall security target model supporting the security target rationale process.

The first issue concerns the identification of threats countered by the security function, e.g. *F.AdminAuth*, related to the administrator authentication activities.
The backwards navigation (horizontal) starts on `AdminAuth:FItem`, goes through the supporting chain (i.e.: `FItem` –> `FunComp` –> `OADTItem` –> `TgrItem`) and ends at threats, which are `associationEnds`:

```
AdminAuth:FItem
self.funComp.ogrGeneric.associationEnds
    ->select(c:TgrGeneric)
```

In the first step – all functional components covered by this `FItem` are identified, in the next step – all security objectives concerning the administrator authentication related to these components, and finally – all threat items covered by these security objectives, though in this case there is only one: `FwlAdminImpers:TDAItem`.
The next issue concerns the answer to the following question: Is there any security function derived from the OSPs? Please note the `isEmpty()` operation applied within the OCL expression, returning the value `TRUE`, because no Pgr generics exist at the association ends of any security objectives in the FW project.

```
AuditFacilities:FItem
isEmpty(self.funComp.ogrGeneric.associationEnds
    ->select(c:PgrGeneric))
```

□

E.6. Remarks on the next step – the TOE development

The main result of the IT security development process is the elaboration of the ST (PP) specification and the related documents implied by the assurance requirements statements. Please note that the defined security functions express the security functionality of the IT product or system. Generally, they can be implemented by applying different rigour expressed by the EAL level. As it was mentioned earlier, the EAL4 level is usually selected for the firewall systems. Very often the given EAL set is augmented by adding any flaw remediation family (ADV_FLR) component, responsible for tracking and correcting flaws made by the developer. Please note the assurance components listed in the Table E.1. To demonstrate that they are satisfied, the developer should provide the right evidence material (see Chapter 11).

The defined security functions represent a particular modules of the FW system (*separation of concerns*), shown in the Fig. E.6. Please note the information flow passing through the firewall administrated by an authorized person.

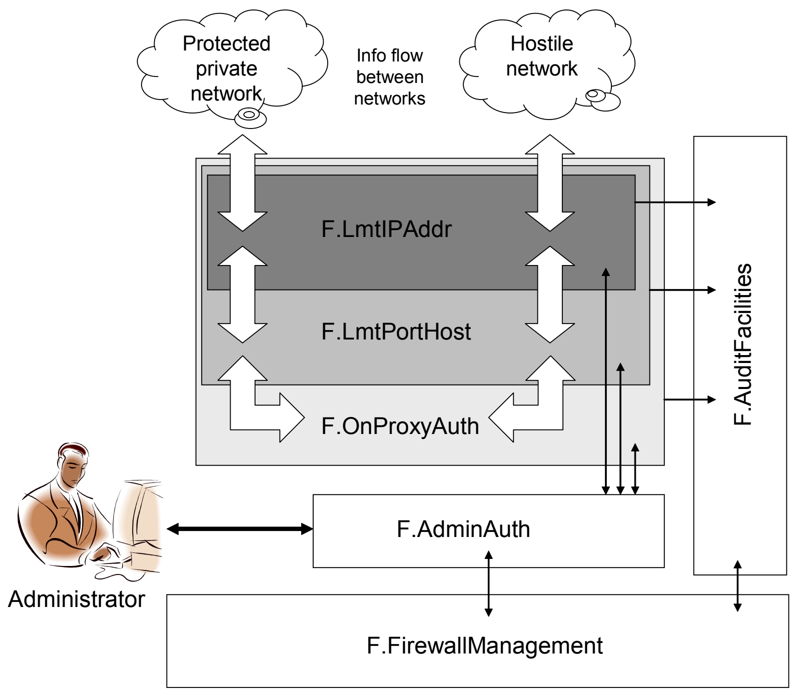


Fig. E.6. Block scheme of the FW system
Rys. E.6. Poglądowy schemat systemu zaporowego FW

The developer ought to satisfy all assurance components by properly arranging and documenting the TOE configuration management, delivery and operation, development, guidance documents, life cycle support, tests and vulnerability assessment. All these issues require a significant effort of the developer. The Example E.13 explains only one of the above issues, i.e. how the considered FW system should be developed.

Example E.13: The FW system (TOE) development requirements.

These requirements are specified by the D-, and C-elements of the chosen components of the ADV class, shown in the Table E.2 (refinement of the Table E.1 with respect to the ADV class components).
□

Table E.2

Requirements dealing with the FW system development [40]

Element (D or C)	Description
ADV_FSP.2 – Fully defined external interfaces	
ADV_FSP.2.1D	The developer shall provide a functional specification.
ADV_FSP.2.1C	The functional specification shall describe the TSF and its external interfaces using an informal style.
ADV_FSP.2.2C	The functional specification shall be internally consistent.
ADV_FSP.2.3C	The functional specification shall describe the purpose and method to use all external TSF interfaces, providing complete details of all effects, exceptions and error messages.
ADV_FSP.2.4C	The functional specification shall completely represent the TSF.

ADV_FSP.2.5C	The functional specification shall include a rationale that the TSF is completely represented.
ADV_HLD.2 – Security enforcing high-level design	
ADV_HLD.2.1D	The developer shall provide the high-level design of the TSF.
ADV_HLD.2.1C	The presentation of the high-level design shall be informal.
ADV_HLD.2.2C	The high-level design shall be internally consistent.
ADV_HLD.2.3C	The high-level design shall describe the structure of the TSF in terms of subsystems.
ADV_HLD.2.4C	The high-level design shall describe the security functionality provided by each subsystem of the TSF.
ADV_HLD.2.5C	The high-level design shall identify any underlying hardware, firmware, and/or software required by the TSF with the presentation of functions provided by the supporting protection mechanisms implemented in that hardware, firmware, or software.
ADV_HLD.2.6C	The high-level design shall identify all interfaces to the subsystems of the TSF.
ADV_HLD.2.7C	The high-level design shall identify which interfaces to the subsystems of the TSF are externally visible.
ADV_HLD.2.8C	The high-level design shall describe the purpose and method to use all interfaces to the subsystems of the TSF, providing details of effects, exceptions and error messages, as appropriate.
ADV_HLD.2.9C	The high-level design shall describe the separation of the TOE into TSP-enforcing and other subsystems.
ADV_IMP.1 – Subset of the implementation of the TSF	
ADV_IMP.1.1D	The developer shall provide the implementation representation for a selected subset of the TSF.
ADV_IMP.1.1C	The implementation representation shall unambiguously define the TSF to a level of detail such that the TSF can be generated without further design decisions.
ADV_IMP.1.2C	The implementation representation shall be internally consistent.
ADV_LLD.1 – Descriptive low-level design	
ADV_LLD.1.1D	The developer shall provide the low-level design of the TSF.
ADV_LLD.1.1C	The presentation of the low-level design shall be informal.
ADV_LLD.1.2C	The low-level design shall be internally consistent.
ADV_LLD.1.3C	The low-level design shall describe the TSF in terms of modules.
ADV_LLD.1.4C	The low-level design shall describe the purpose of each module.
ADV_LLD.1.5C	The low-level design shall define the interrelationships between the modules in terms of provided security functionality and dependencies on other modules.
ADV_LLD.1.6C	The low-level design shall describe how each TSP-enforcing function is provided.
ADV_LLD.1.7C	The low-level design shall identify all interfaces to the modules of the TSF.
ADV_LLD.1.8C	The low-level design shall identify which interfaces to the modules of the TSF are externally visible.
ADV_LLD.1.9C	The low-level design shall describe the purpose and method to use

	all interfaces to the modules of the TSF, providing details of effects, exceptions and error messages, as appropriate.
ADV_LLD.1.10C	The low-level design shall describe the separation of the TOE into TSP-enforcing and other modules.
ADV_RCR.1 – Informal correspondence demonstration	
ADV_RCR.1.1D	The developer shall provide an analysis of correspondence between all adjacent pairs of TSF representations that are provided.
ADV_RCR.1.1C	For each adjacent pair of provided TSF representations, the analysis shall demonstrate that all relevant security functionalities of the more abstract TSF representation are correctly and completely refined in the less abstract TSF representation.
ADV_SPM.1 – Informal TOE security policy model	
ADV_SPM.1.1D	The developer shall provide a TSP model.
ADV_SPM.1.2D	The developer shall demonstrate correspondence between the functional specification and the TSP model.
ADV_SPM.1.1C	The TSP model shall be informal.
ADV_SPM.1.2C	The TSP model shall describe the rules and characteristics of all policies of the TSP that can be modelled.
ADV_SPM.1.3C	The TSP model shall include a rationale that demonstrates that it is consistent and complete with respect to all policies of the TSP that can be modelled.
ADV_SPM.1.4C	The demonstration of correspondence between the TSP model and the functional specification shall show that all security functions in the functional specification are consistent and complete with respect to the TSP model.

REFERENCES

1. ACSA: http://www.acsac.org/waepssd.
2. Apted A.J., Carthigaser M., Lowe Ch.: Common Problems with the Common Criteria, Proceedings of the 3rd International Common Criteria Conference, May 2002.
3. AUTOFOCUS: http://autofocus.informatik.tu-muenchen.de.
4. AOSD: http://www.aosd.net/.
5. Białas A.: Wprowadzenie do problematyki projektowania i oceny zabezpieczeń teleinformatycznych, Studia Informatica vol. 22, Number 1(43), Silesian University of Technology Press, Gliwice 2001, pp. 263÷287 („*Introduction to IT security development and evaluation*", in Polish).
6. Białas A.: Modelowanie i ocena zabezpieczeń teleinformatycznych, Studia Informatica vol. 23, Number 2B(49), Silesian University of Technology Press, Gliwice 2002, pp. 219÷232 („*Security modelling and evaluation*", in Polish).
7. Białas A.: Sposób formalnego wyrażania własności bezpieczeństwa teleinformatycznego, Studia Informatica vol. 24, Number 2B(54), Silesian University of Technology Press, Gliwice 2003, pp. 265÷278 („*Formal description of the security features"*, in Polish).
8. Białas A.: Hierarchy of the Assets Model for the Information Technology Security Management, Archiwum Informatyki Teoretycznej i Stosowanej, Polska Akademia Nauk, vol. 15 (2003), z. 2, 2003, pp. 109÷120.
9. Białas A.: The automated support for the information and communications technology security management, Elektronnoje Modelirovanije, vol. 25, No. 4, Ukrainian National Academy of Sciences, 2003, pp. 39÷50.
10. Białas A.: Modelowanie zasobów teleinformatycznych oraz funkcji zabezpieczających według Wspólnych Kryteriów, Rozdział w: Grzywak A., Kwiecień A. (redakcja): Współczesne problemy sieci komputerowych – zastosowanie i bezpieczeństwo, Wydawnictwa Naukowo-Techniczne, 2004, pp. 351÷368. (*"ICT Assets and security functions modelling – Common Criteria approach"*, in Polish).

11. Białas A.: The Assets Inventory for the Information and Communication Technologies Security Management, Archiwum Informatyki Teoretycznej i Stosowanej, Polska Akademia Nauk, vol.16 (2004), z. 2, 2004, pp. 93÷108.
12. Białas A.: Bezpieczeństwo teleinformatyki – wzorcowa praktyka czy miara gwarantowana, Rozdział w: Marecki F., Grabara J.K., Nowak J.S. (red.): Systemy informatyczne – bankowość i finanse, Wydawnictwa Naukowo-Techniczne, 2004, pp. 323÷364, (*„Information security management and IT security evaluation”*, in Polish).
13. Białas A.: IT security development – computer-aided tool supporting design and evaluation, In: Kowalik J, Górski J., Sachenko A. (editors): Cyberspace Security and Defense: Research Issues, NATO reference: ARW 980492, NATO Science Series II, vol. 196, Springer, Dordrecht, 2005, pp. 3÷23.
14. Białas A.: Identifying the features of the IT security-related products for the IT development process according to Common Criteria, Archiwum Informatyki Teoretycznej i Stosowanej, Polska Akademia Nauk, vol. 17 (2005), z. 1, 2005, pp. 3÷18.
15. Białas A.: IT security modelling, In: Arabnia, H. R., Editor; Liwen He & Youngsong Mun, Associate Co Editors, Proceedings of the 2005 International Conference on Security and Management (The World Congress In Applied Computing – SAM'05: June, Las Vegas, USA), ISBN# 1 932415 82 3, Publisher: CSREA Press, 2005, pp. 502÷505.
16. Białas.: Critical information infrastructure protection – research issues and activities, In: Stepnowski A (Editor), Ruciński A.& Kosmowski K. (Co-Editors), Proceedings of the IEEE International Conference on Technologies for Homeland Security and Safety – TEHOSS’2005, Gdansk, September 28-30, 2005, ISBN 83-917681-9-8, pp. 369÷374.
17. Białas A.: The ISMS Business Environment Elaboration Using a UML Approach, In: Zieliński K., Szmuc T. (editors): Software Engineering: Evolution and Emerging Technologies, IOS Press, Amsterdam, 2005, ISBN: 1 58603-559-2, pp. 99÷110.
18. Białas A.: A UML approach in the ISMS implementation, In: Dowland P., Furnell S., Thuraisingham B., Wang X.S. (eds): Security management, integrity, and internal control in information systems, IFIP TC-11 WG 11.1 & WG 11.5 Joint Working Conf., Springer Science + Business Media, New York 2005, ISBN-10:0-387-29826-6, pp. 285÷297.
19. Białas A.: Wspólne kryteria do projektowania i oceny zabezpieczeń, Szkolenie dla Centrum Analiz Kryptograficznych i Bezpieczeństwa Teleinformatycznego MON, Warszawa, 29-30 listopada 2005 (*”Common Criteria for IT security development and evaluation – Training handbook for IT security labs”*, in Polish).
20. Białas A.: Information security systems vs. critical information infrastructure protection systems similarities and differences. In: Zamojski W., Mazurkiewicz J., Sugier J., Walkowiak T.: Proceedings of the International Conference on Dependability of Computer

Systems DepCoS-RELCOMEX, May 2006, IEEE Computer Society Los Alamitos, Washington, Tokyo, 2006, ISBN 0-7695-2565-2, pp. 60÷67.

21. Białas A.: Using ISMS concept for critical information infrastructure protection. In: Balducelli A., Bologna S. (eds), Proceedings of the International Workshop on "Complex Network and Infrastructure Protection – CNIP'06", Italian National Agency for New Technologies, Energy and the Environment (ENEA), Rome, March 28-29, 2006, pp. 415÷426, http://ciip.casaccia.enea.it/cnip06
22. Białas A.: A semiformal approach to the security problem of the target of evaluation (TOE) modeling, In: Arabnia, H. R., Aissi S. (Editors), Vert G. L., Williams P.A.H. (Associate Co Editors), Proceedings of the 2006 International Conference on Security and Management (The World Congress In Applied Computing – SAM'06: June, Las Vegas, USA), ISBN# 1-60132-001-9, Publisher: CSREA Press, 2006, pp. 19÷25.
23. Białas A.: Bezpieczeństwo informacji i usług w nowoczesnej instytucji i firmie, Wydawnictwa Naukowo-Techniczne, Warszawa 2006, 2007, ISBN 83-204-3155-7 (*„Information security within modern organizations and companies",* in Polish).
24. Białas A.: Półformalna reprezentacja procesu projektowania zabezpieczeń tele-informatycznych, Rozdział w: Pochopień B., Kwiecień A., Grzywak A., Klamka J. (redakcja): Nowe technologie sieci komputerowych, Wydawnictwa Komunikacji i Łączności, 2006, pp. 329÷336 (*„Semiformal approach to the IT security development process",* in Polish).
25. Białas A.: Development of an Integrated, Risk-based Platform for Information and E-services Security, In: Górski J.: Computer Safety, Reliability, and Security, 25th International Conference SAFECOMP2006, Lecture Notes in Computer Science (LNCS4166), Springer Verlag Berlin Heidelberg New York 2006, pp. 316÷329.
26. Białas A.: Specification of security environment of IT security-related products according to Common Criteria, Theoretical and Applied Informatics, ISSN 1896-5334, vol. 18 (2006) z. 2. pp. 141÷157.
27. Białas A.: Konstruowanie zabezpieczeń teleinformatycznych zgodnie ze standardem ISO/IEC 15408 – Common Criteria, „II Functional Safety Management Conference", Jurata, October 2007. (*„CC-compliant IT security development process",* in Polish).
28. Bialas A.: Semiformal framework for ICT security development, The 8th International Common Criteria Conference, Rome, 25-27 September 2007.
29. Białas A.: Modeling the Security Objectives According to the Common Criteria Methodology, In: Aissi S., Arabnia H. R. (Editors), Daimi K., Gligoroski D., Markowsky G., Solo A.M.G. (Associate Co Editors), Proc. of the 2007 International Conference on Security and Management (The World Congress In Applied Computing – SAM'07: June, Las Vegas, USA), ISBN# 1-60132-048-5, 2007, Publisher: CSREA Press, pp. 223÷229.

30. Białas A.: Semiformal Approach to the IT Security Development In: Zamojski W., Mazurkiewicz J., Sugier J., Walkowiak T.: Proceedings of the International Conference on Dependability of Computer Systems DepCoS-RELCOMEX 2007, IEEE Computer Society, Los Alamitos, Washington, Tokyo, ISBN 0-7695-2850-3, pp. 3÷11.
31. Białas A., Lisek K.: Integrated, Business-Oriented, Two-Stage Risk Analysis, Journal of Information Assurance and Security (JIAS), vol. 2, issue 3, September 2007, www.dynamicpublishers.com/JIAS
32. Białas A.: Szkieletowy system konstruowania zabezpieczeń teleinformatycznych – przegląd i wyniki prac, Rozdział w: Kwiecień A., Ober J., Pochopień B., Gaj P. (redakcja): Sieci Komputerowe, Tom2 Aplikacje i Zastosowania, Wydawnictwa Komunikacji i Łączności, 2007, pp. 311÷320 (*IT security development framework – a project overview and results*, in Polish).
33. Białas A.: Advanced IT Security Development Process – through Enhancement of IT Security Development Process to Better Assurance, Chapter 13 in monograph: Kosmowski K.T. (Ed): Functional Safety Management In Critical Systems, Politechnika Gdańska, Fundacja Rozwoju Uniwersytetu Gdańskiego, Gdańsk 2007 (ISBN 978-83-7531-006-1).
34. B-Method/Tools: http://www.b-core.com
35. den Braber F, Lund S., Stølen K.: Using the CORAS Threat Modelling Language to Document Threat Scenarios for several Microsoft relevant Technologies, Report STF90 A04057, Sintef, 2004.
36. Booch G., Rumbaugh J., Jacobson I.: UML- Przewodnik użytkownika, Wyd. II, Wydawnictwa Naukowo-Techniczne, Warszawa 2002, (*"The Unified Modeling Language – User Guide"*).
37. Cakir M.: Evaluation of organizational information systems according to CC and ISO 17799, 5th International CC Conference, Berlin, September 2004.
38. ISO/IEC 15408-1, Information technology – Security techniques – Evaluation criteria for IT security – Introduction and general model (*Common Criteria Part 1*).
39. ISO/IEC 15408-2, Information technology – Security techniques – Evaluation criteria for IT security – Security functional requirements (*Common Criteria Part 2*).
40. ISO/IEC 15408-3, Information technology – Security techniques – Evaluation criteria for IT security – Security assurance requirements (*Common Criteria Part 3*).
41. Common Criteria Evaluation and Validation Scheme for Information Technology Security, Organization Management and Concept of Operation, v.2.0., NIST – NSA.
42. Common Criteria portal: http://www.commoncriteriaportal.org/
43. CCToolbox: http://cc-control.sparta.com/

44. Cheesman J., Daniels J.: Komponenty w UML, Wydawnictwa Naukowo-Techniczne, Warszawa 2004, (*"UML Components – A Simple Process for Specifying Component-Based Software"*).
45. Common Evaluation Methodology for Information Technology Security, Part 1: Introduction and General Model.
46. Common Evaluation Methodology for Information Technology Security, Part 2: Evaluation Methodology.
47. Chapman R.: SPARK – a state-of-the-practice approach to the Common Criteria implementation requirements, 2nd International CC Conference, Brighton, July 2001.
48. CI^{2}RCO: www.ci2rco.org
49. Cockburn A.: Jak pisać efektywne przypadki użycia?, Wydawnictwa Naukowo-Techniczne, Warszawa 2004, (*"Writing Effective Use Cases"*).
50. Ekelhart A., Fenz, S., Goluch, G., and Weippl, E.: Ontological Mapping of Common Criteria's Security Assurance Requirements, 2007 IFIP, Volume 232, New Approaches for Security, Privacy and Trust in Complex Environments, eds. Venter, H-, Eloff, M-, Labuschagne, L., Eloff, J., von Solms, R., (Boston: Springer), pp. 85÷95.
51. Galitzer S.: Introducing Engineered Composition (EC): An Approach for Extending the Common Criteria to Better Support Composing Systems, Published in the Workshop for Application of Engineering Principles to System Security Design (WAEPSSD) Proceedings, September 2003.
52. Górski J.: Trust case – A case for trustworthiness of IT infrastructures, In: Kowalik J, Górski J., Sachenko A. (editors): Cyberspace Security and Defense: Research Issues, NATO reference: ARW 980492, NATO Science Series II, vol. 196, Springer, Dordrecht, 2005, pp. 125÷141.
53. Hays D.: Security Engineering: Science or Art?, Published in the Workshop for Application of Engineering Principles to System Security Design (WAEPSSD) Proceedings, September 2003.
54. Hunstad A., Hallberg J.: Design for securability – Applying engineering principles to the design of security architecture, Published in the Workshop for Application of Engineering Principles to System Security Design (WAEPSSD) Proceedings, September 2003.
55. Hall-May M., Kelly T.: Using Agent-Based Modelling Approaches to Support the Development of Safety Policy for System of Systems, In: Górski J.: Computer Safety, Reliability, and Security, 25th International Conference SAFECOMP2006, Springer Lecture Notes in Computer Science (LNCS4166), Springer Verlag Berlin Heidelberg New York 2006, ISBN 3-540-45762-3, pp. 330÷343.
56. Hwa-Jong S.: Development and utilization of automatic generation tool for evaluation report, 5th International CC Conference, Berlin, September 2004.

57. ICCC: http://www.expotrack.com/iccc/english/proceedings.asp
58. ISO 27001:2005 Information security management systems – Specification with guidance for use.
59. ISO/IEC TR 15443, Information technology – Security techniques – A framework for IT security assurance.
60. ISO/IEC TR 15446:2004, Information technology – Security techniques – Guide for the production of protection profiles and security targets.
61. Information Technology Security Evaluation Criteria (ITSEC), EGKS-EWG-EAG, Bruessel, Juni 1991.
62. Jürjens J., Houmb S.H.: Risk-driven development of security-critical systems using UMLsec, LADC 2003, São Paulo, Oct. 21-24, 2003.
63. Jung-Shian Li: Development of CC in Taiwan, 5th International CC Conference, Berlin, September 2004.
64. Jürjens J.: Developing Secure Systems with UMLsec – From Business Processes to Implementation, VIS 2001, Kiel (Germany), 12-14 Sept. 2001, Vieweg-Verlag, 2001.
65. Jürjens J., Secure Systems Development with UML - Applications to Telemedicine, CORAS workshop, Int. Conf. on Telemedicine (ICT2002), Regensburg, September, 2002.
66. Jürjens J.: UMLsec: Extending UML for Secure Systems Development, UML 2002, Dresden, LNCS, Springer-Verlag, 2002.
67. Jürjens J.: A UML statecharts semantics with message-passing, Symposium of Applied Computing (SAC 2002), Madrid, March 10-14, ACM, 2002.
68. Jürjens J.: Using UMLsec and Goal-Trees for Secure Systems Development, Symposium of Applied Computing (SAC 2002), Madrid, March 10-14, ACM, 2002.
69. Jürjens J.: Formal Semantics for Interacting UML subsystems, IFIP TC6/WG6.1 Fifth International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS 2002), Twente, March 20-22, Kluwer, 2002.
70. Jürjens J., Model-based Security with UMLsec, UML Forum, Tokyo, Apr. 17, 2003.
71. Jürjens J.: Formal Development of Critical Systems with UML, ETAPS 03, European joint conferences on Theory And Practice of Software 2003, Warschau, April 2003.
72. Jürjens J.: Secure Systems Development with UML, Springer-Verlag, 2005.
73. Krueger B.: Application of the Common Criteria to Information Security Management Systems – A study, 5th International CC Conference, Berlin, September 2004.
74. Lavatelli C.: EDEN: A formal framework for high level security CC evaluations, e-Smart' 2004, Sophia Antipolis 2004.

75. Ling R., Latapie H., Tran V.: Expressing Common Criteria Security Requirements in Domain Models in Model-based Architecture, 6th Annual Workshop on Distributed Objects and Component Security 2002.
76. Leffingwell D., Widrig D.: Zarządzanie wymaganiami, Wydawnictwa Naukowo-Techniczne, Warszawa 2003, (*Managing Software Requirements – A Unified Approac*h).
77. Melton R.: Integration of risk management with the Common Criteria (ISO/IEC 15408:1999), 5th International CC Conference, Berlin, September 2004.
78. Motre S., Teri C.: Using Formal and Semiformal Methods for a Common Criteria Evaluation, In: Marting L (Ed): EuroSmart Security Conference, Marseille, slide version, pp. 337÷349, 2000.
79. Murray W.H.: Position paper for the Workshop for Application of Engineering Principles to System Security Design, Published in the Workshop for Application of Engineering Principles to System Security Design (WAEPSSD) Proc., September 2003.
80. Menezes A., van Oorschot P., Vanstone S.: Handbook of Applied Cryptography, CRC Press, 1996, ver. downloaded in 2002 from: http://www.cacr.math.uwaterloo.ca/hac
81. Naaman N.: A unified framework for information assurance, 5th International CC Conference, Berlin, September 2004.
82. Nash M.: Simpler security targets, 5th International CC Conference, Berlin, Sept. 2004.
83. UML 2.0 OCL Specification, OMG, 2003.
84. Oltra M.A.: Security Framework (draft 0.1.2), ITEA – Osmose, 2006 (WP2-031023-1), available at: http://www.itea-osmose.org.
85. Object Modelling Group portal: http://www.omg.org.
86. Pattinson F.: BS 7799-2 and Common Criteria – Supporting the business of software development, 5th International CC Conference, Berlin, September 2004.
87. Security Target BSI-DSZ-CC-0153: First Evaluation of Philips P8WE5032 Secure 8-bit Smart Card Controller, Philips Semiconductors Hamburg, September 1999.
88. Certification Report BSI-DSZ-CC-0153-1999 for Philips Smart Card Controller P8WE5032V0B from Philips Semiconductors Hamburg, BSI, November 1999.
89. Short form specification – Philips P8WE5032 Secure 8-bit Smart Card Controller, Philips Semiconductors, rev.1.0, July 2000.
90. POZIT: Białas A. Praca zbiorowa pod red.: Metodyka prowadzenia badań i oceny środków teleinformatycznych, Projekt celowy KBN pt. System wspomagania projektowania i oceny zabezpieczeń teleinformatycznych, Instytut Systemów Sterowania, 2004 (target project reports: „*IT security development and evaluation*" – in Polish).
91. UML 2.0 OCL Specification, Appendix A: Semantics, 2003, available at: www.omg.org, developed on the basis of: Richters M.: A precise approach to validating UML models and

OCL constraints. Ph.D thesis, Universitaet Bremen, Logos Verlag, Berlin, BISS Monographs, No.14, 2002.

92. Robinson K.: An Introduction to the B Method – An Overview, School of Computer Science & Engineering, 2003.
93. SecCert: http://www.cbst.iss.pl .
94. SecCert Users Guide, Instytut Systemów Sterowania, 2006.
95. SecOffice: http://www.cbst.iss.pl .
96. SECOQC: http://www.secoqc.net/.
97. Spafford E.H.: Exploring Common Criteria: Can it Ensure that the Federal Government Gets Needed Security in Software, Published in the Workshop for Application of Engineering Principles to System Security Design (WAEPSSD) Proceedings, Sept. 2003.
98. SPARK: http://praxis-cs.co.uk/sparkada/publications.asp.
99. Stoneburner G.: Underlying Technical Models for Information Technology Security, NIST Special Publication, Gaithersburgh 2001.
100. TL FIT: http://trusted-logic.fr.
101. TL SET: http://trusted-logic.fr.
102. UML: http://www.omg.org/uml/.
103. Warmer J., Kleppe A.: OCL – Precyzyjne modelowanie w UML, Wydawnictwa Naukowo-Techniczne, Warszawa 2003, (*The Object Constraint Language – Precise Modeling with UML*).
104. Win De B., Piessens F., Joosen W.: On the importance of the separation-of-concerns principle in secure software engineering, Published in the Workshop for Application of Engineering Principles to System Security Design (WAEPSSD) Proc., September 2003.
105. Yavagal D.S., Lee S.W., Ahn G-J., Gandhi R.A.: Common Criteria Requirements Modeling and its Uses for Quality of Information Assurance (QoIA), In: Proc. of the 43rd Annual ACM Southeast Conference (ACMSE '05), Vol. 2, pp. 130÷135, March 18-20, Kennesaw State Univ. Kennesaw, Georgia. 2005.
106. Białas A.: Ontology-based Approach to the Common Criteria Compliant IT Security Development, In: Proceedings of the 2008 International Conference on Security and Management (The World Congress In Applied Computing – SAM'08), July 2008, Las Vegas, USA (accepted).

SEMIFORMAL COMMON CRITERIA COMPLIANT IT SECURITY DEVELOPMENT FRAMEWORK

Keywords: assurance, Common Criteria, computer-aided, design, development, evaluation, formal method, framework, IT security, UML, OCL, modelling, security engineering, semiformal method

Abstract

The monograph presents an IT Security Development Framework (ITSDF) based on the Common Criteria (ISO/IEC 15408) family of standards for the product designers and evaluators. The system, compliant with ISO/IEC TR 15446, is based on the enhanced concept of generics, advanced functionality, recent information security management standards, and risk analysis.

A computer-aided tool was developed with the use of the UML-based (*Unified Modelling Language*) framework presented there. Due to the semiformal character of the Common Criteria and the UML methodologies, the framework has a semiformal character too. The formal, OCL-based (*Object Constraint Language*) method elements were introduced for the selected areas of the framework, where they can bring real advantages, especially to improve the specification means.

The introductory part of the monograph presents the key term, i.e. the assurance, and general Common Criteria approach to provide the assurance of the IT product or systems. Rigorous IT product or system development to reach assurance is very complicated due to many details, feedbacks, auxiliary analysis and step-by-step rationale. The development is laborious and requires expert knowledge. These difficulties are seen as a barrier to the dissemination of IT solutions of the higher assurance. The motivation of the work presented in the monograph is how to improve the IT security development process, i.e. how to perform it more precisely, formally, consistently, and more effectively, i.e. more quickly and cheaply. The proposed solution to the problem is based on the UML/OCL approach, hence selected issues dealing with the application of this methodology in the information security domain are provided. At the end of the introductory chapter the general concept of the UML/OCL framework for IT security development is summarized together with the means and ways to develop this framework.

The second chapter presents general background of the elaborated IT Security Development Framework and its software implementation. At the beginning, the current state of technology was reviewed, with focus on the gaps identification. The review includes the most relevant publications on: IT security development process, general modelling aspects, UML and OCL implementation, semiformal and security engineering methods, particularly concerning Common Criteria, and computer-aided tools. The relationships between IT product or system development process and their IT security development process is discussed. Additionally, this chapter gives a summary of the Common Criteria development process, identifying points whose support for the developers is especially required (i.e. developers' needs) and, generally, presents the concept of the developed framework (main structural model and the state machine representing its elaboration). Short discussion of the available specification means is added, that is the motivation to improve them. The concept presented in the monograph, dealing with the elaboration of the IT Security Development Framework, encompasses two basic issues:

- creating the means to build the security specifications, i.e. "a security property language",
- workout of the semiformal (UML/OCL-based) model of this development process.

The first issue is discussed in the chapter 3, the second in chapters 4-10, each related to one section of the CC-defined Security Target (ST) presenting the IT product/system security requirements.

Common Criteria include semiformal specification means, i.e. components only to specify the functional and assurance security requirements. Due to the absence of adequate specification means for other IT security development stages (the TOE security environment, TOE security objectives, security requirements for the environment, security functions), the monograph provides the set of enhanced generics which, together with the above mentioned components, constitutes a coherent set of the means, i.e. a language to carry out security specifications (ST). The common, UML/OCL-based model for these means is assumed. This creates quite new possibilities for developers, making the development process easier and more precise, e.g. operation on specification elements, decision support, coverage analysis, risk analysis, better compliance with the information security management, etc.

Chapters 4 to 10 encompass the whole IT security development process compliant with the Security Target structure. Instead of commonly used informal textual descriptors, the UML models, supported by the OCL, were applied to specify this framework. The initial stage of the development methodology focuses on capturing the basic features of an IT product or system as "input data". The ST introduction is elaborated, presenting general features of the IT product or system, called there Target of Evaluation (TOE). On this basis the TOE security environment, presenting threats, security policy rules and assumptions, is worked out. The security objectives covering the identified security problems are elaborated with the use of the TOE security environment specification. The security objectives are used to specify the security requirements – functional and assurance. The functional security requirements allow to define the security functions which are implemented at the EAL-described (EAL1-EAL7) rigour level. A very important issue is the rationale processes which is obligatory between development stages.

The monograph concerns development, hence the TOE evaluation against the declared EAL is presented there in a concise way in the chapter 11. The evaluation model compliant with the Common Criteria methodology is presented.

Although the monograph is focused on the theoretical aspects of the ITSDF framework, please note that this framework was implemented as the computer-aided tool both for the development and evaluation. The author, referring for details to technical documentation and the demo version of the software tool, shows basic implemented issues, like the design library containing hundreds of specification means, wizard providing step-by-step assistance to developers, design support by the predefined supporting chains, visualization facilities, automatically generated ST, design evaluation, etc.

The final chapter, 13, summarizes the objectives, range and results of the whole work. The five appendices of the monograph include: the basic Common Criteria terminology, selected OCL syntax and semantics used for the UML model representation in the monograph, selected cryptographic terms concerning the use of the UMLsec approach, basic principles of naming the terms, and an example illustrating the developed methodology with the use of a firewall design.

PÓŁFORMALNY SYSTEM SZKIELETOWY DO KONSTRUOWANIA ZABEZPIECZEŃ INFORMATYCZNYCH ZGODNY Z METODYKĄ WSPÓLNE KRYTERIA

Słowa kluczowe: uzasadnione zaufanie, Common Criteria, Wspólne kryteria, wspomaganie komputerowe, konstruowanie, prace rozwojowe, ocena zabezpieczeń, metody formalne, zrąb, system szkieletowy, bezpieczeństwo informacji, UML, OCL, modelowanie, inżynieria zabezpieczeń, metody półformalne

Streszczenie

Monografia dotyczy zagadnienia konstruowania zabezpieczeń wbudowywanych w produkty lub systemy informatyczne (IT) z zamiarem poddania niezależnej ocenie tych zabezpieczeń, zgodnie ze standardem ISO/IEC 15408 „Wspólne kryteria oceny zabezpieczeń teleinformatycznych" (*Common Criteria*). Dzięki specjalnej, rygorystycznej metodzie konstruowania, a potem niezależnej ocenie zabezpieczeń, poświadczanej certyfikatem, można dyskutować o uzasadnionym zaufaniu do zabezpieczeń (*assurance*). Jest ono mierzalne z wykorzystaniem tak zwanych poziomów uzasadnionego zaufania (*Evaluation Assurance Levels*), w zakresie od EAL1 do EAL7. Przypomnijmy, że pierwsza cześć wspomnianej normy prezentuje informacje ogólne – „filozofię" kreowania uzasadnionego zaufania do zabezpieczeń, druga zawiera katalog komponentów (elementarnych wymagań) funkcjonalnych, służących do modelowania zachowania zabezpieczeń, trzecia zaś – katalog komponentów opisujących uzasadnione zaufanie. Przedstawiana problematyka dotyczy bardzo szerokiego spektrum produktów informatycznych (sprzętu, oprogramowania, w tym układowego) oraz zbudowanych z nich systemów – określanych wspólnym mianem przedmiotów oceny (*TOE – Target of Evaluation*).

Na tym tle niniejsza monografia prezentuje półformalny system szkieletowy do konstruowania zabezpieczeń informatycznych według ISO/IEC 15408, przeznaczony dla konstruktorów informatyków i elektroników oraz specjalistów oceniających zabezpieczenia. System szkieletowy zawiera model procesów konstruowania (struktury danych i działania) oraz modele środków do specyfikowania tworzonych projektów. Oferuje rozbudowane możliwości

dla konstruktorów w zakresie wspomagania projektowania, w tym prowadzenia analizy rozwiązań, wspomagania decyzji oraz analizy ryzyka. Zachowana została zgodność ze standardami zarządzania bezpieczeństwem informacji w zakresie definiowania reguł polityki bezpieczeństwa. System opiera się również na wprowadzonej koncepcji tak zwanych zaawansowanych (rozszerzonych) generyków, wykorzystywanych obok komponentów funkcjonalnych i uzasadniających zaufanie do specyfikowania projektów zabezpieczeń.

System szkieletowy opracowano, stosując szeroko język UML (*Unified Modelling LANguage*), co w rezultacie pozwoliło dokonać implementacji tych modeli i opracować komputerowe narzędzie wspomagające pracę konstruktorów. Zarówno Wspólne kryteria, jak i UML zaliczane są do metod półformalnych, stąd opracowany system szkieletowy posiada również taki charakter, aczkolwiek w sposób rozważny i z uwzględnieniem potencjalnych korzyści pewne jego elementy przedstawiono w sposób formalny, opierając się głównie na języku OCL (*Object Constraint Language*) oraz prostych operacjach na zbiorach.

Rozdział 1., wprowadzający do monografii, wyjaśnia kluczowe pojęcie, jakim jest uzasadnione zaufanie oraz przedstawia, w świetle standardu Wspólne kryteria, w jaki sposób uzyskuje się uzasadnione zaufanie dla produktu lub systemu informatycznego podczas jego konstruowania, oceny i eksploatacji. Jest to trudne zadanie ze względu na konieczność zapanowania nad wieloma detalami, wzajemnymi zależnościami i sprzężeniami zwrotnymi w toku rygorystycznego procesu konstruowania. Pamiętajmy bowiem, że im większy stosuje się rygoryzm konstruowania, dokumentowania, testowania, analiz zachowania itp., tym wyższe może być uzasadnione zaufanie. Opracowywanie tego typu produktów lub systemów jest pracochłonne i wymaga specjalistycznej wiedzy, a trudności z tym związane są ciągle uznawane za barierę w ich upowszechnieniu. Motywacją do podjęcia pracy, której wyniki przedstawia niniejsza monografia, był zamiar udoskonalenia procesu konstruowania zabezpieczeń, tak by przez głębszą formalizację tego procesu poprawić dokładność i spójność projektów oraz obniżyć koszt i czas ich realizacji. W tym celu wykorzystano możliwości, jakie stwarza metodyka UML/OCL.

Rozdział 2. przedstawia szeroko podstawy prowadzonych prac rozwojowych nad systemem szkieletowym, w tym jego implementacją w postaci oprogramowania wspomagającego działania konstruktorów. Dokonano przeglądu publikacji oraz istniejących rozwiązań technologicznych w tym zakresie (inżynieria zabezpieczeń, metody oparte na UML i ich rozszerzenia, metody i narzędzia wspomagające konstruowanie zabezpieczeń według Wspólnych kryteriów). Przedstawiono związki między procesem konstruowania produktu lub systemu a procesem konstruowania dla nich zabezpieczeń. Niniejsza praca dotyczy drugiego zagadnienia, aczkolwiek wymagało to również krótkiego wprowadzenia do pierwszego z nich, a także zwięzłego przedstawienia oceny zabezpieczeń, gdyż każda z tych kwestii decyduje o osiągnięciu uzasadnionego zaufania. Rozdział opisuje krótko proces konstruowania zabezpieczeń, potrzeby konstruktorów w zakresie wspomagania oraz ogólną koncepcję systemu szkieletowego w postaci struktury i maszyny stanowej. Koncepcja ta została rozwinięta w kolejnych rozdziałach pracy. Przedstawiona w monografii myśl dotyka dwóch podstawowych zagadnień:

- opracowania środków (języka) do specyfikowania własności bezpieczeństwa produktów lub systemów informatycznych, zwanych przedmiotami oceny (TOE),
- opracowania modelu procesu konstruowania zabezpieczeń z wykorzystaniem UML/OCL.

Pierwsze zagadnienie zostało przedstawione obszernie w rozdziale 3, w rozdziałach od 4. do 10. pokazano zaś proces konstruowania zabezpieczeń informatycznych wyrażony w języku UML/OCL. Rozdziały od 4. do 10. odpowiadają strukturze dokumentu pt. „Zadanie zabezpieczeń" (*ST – Security Target*), określonego w standardzie, stąd są one różnej długości.

Jak już wspomniano, Wspólne kryteria dostarczają półformalnych środków specyfikowania w postaci komponentów reprezentujących elementarne wymagania funkcjonalne albo uzasadniające zaufanie. Ze względu na brak odpowiednich środków specyfikowania dla pozostałych (innych niż wymagania bezpieczeństwa) etapów konstruowania (otoczenie zabezpieczeń, cele zabezpieczeń, wymagania otoczenia, funkcje zabezpieczające), w monografii wypełniono tę lukę oraz zdefiniowano zbiór tak zwanych zaawansowanych generyków i w ten sposób uzyskano kompletny oraz jednolity zestaw elementów półformalnych do tworzenia specyfikacji. Wykorzystano do tego celu właściwości języków UML i OCL. Stworzyło to całkiem nowe możliwości dla konstruktorów (elementy wspomagania decyzji podczas projektowania, analiza pokrycia elementów specyfikacji, analiza ryzyka, poprawa zgodności ze standardami zarządzania bezpieczeństwem informacji w zakresie reguł polityki bezpieczeństwa itp.) oraz umożliwiło późniejszą implementację programową modeli generyków i komponentów w postaci biblioteki projektowej.

Rozdziały od 4. do 10. opisują całościowo proces konstruowania zabezpieczeń informatycznych, sprowadzający się do wypełnienia treścią struktury zadania zabezpieczeń (ST). Zamiast nieformalnych, mniej precyzyjnych określeń słownych, do budowy specyfikacji zastosowano elementy zamodelowane w UML ze wsparciem OCL. W pierwszym kroku metodyki konstruowania uwagę skupiono na zidentyfikowaniu podstawowych cech produktu lub systemu informatycznego, jako danych wejściowych dla kolejnych kroków postępowania. W rezultacie powstała specyfikacja wprowadzenia do ST, określająca podstawowe cechy przedmiotu oceny (TOE). Na tej podstawie, przy pewnych założeniach dotyczących TOE i jego środowiska, analizowane są zagrożenia oraz ustalane są reguły polityki bezpieczeństwa TOE, co prowadzi do wypracowania specyfikacji otoczenia zabezpieczeń przedmiotu oceny (w nowszej wersji standardu zwanej określeniem problemu

bezpieczeństwa) jako kolejnej sekcji ST. Zidentyfikowane problemy (zagrożenia, reguły polityki, założenia) należy teraz pokryć ich rozwiązaniami, czyli celami zabezpieczeń dla TOE i jego środowiska eksploatacyjnego, w wyniku czego powstanie specyfikacja celów zabezpieczeń. Z kolei te cele należy przetransformować w bardziej sformalizowaną postać, to znaczy w wymagania bezpieczeństwa (funkcjonalne i uzasadnienia zaufania) dla TOE i jego środowiska. Wymagania co do uzasadnienia zaufania do TOE wynikają głównie z poziomu uzasadnionego zaufania (EAL) zadeklarowanego dla przedmiotu oceny. Wymagania funkcjonalne dla TOE są transformowane na funkcje zabezpieczające, wymagania uzasadniające zaufanie decydują zaś o stopniu rygoryzmu stosowanego przy implementowaniu tych funkcji w konkretnym produkcie lub systemie informatycznym. Niezwykle istotnymi, a zarazem trudnymi dla konstruktorrów, są procesy uzasadniania transformacji jednej specyfikacji w drugą, to znaczy uzasadniania: celów zabezpieczeń opracowanych na podstawie otoczenia, wymagań bezpieczeństwa wyprowadzonych z tych celów oraz funkcji zabezpieczających, wynikających z wymagań funkcjonalnych.

Monografia dotyczy konstruowania zabezpieczeń, stąd zagadnienie ich oceny z uwzględnieniem deklarowanego poziomu EAL przedstawiono w rozdziale 11. dość skrótowo, ograniczając się tylko do modeli ogólnych, aczkolwiek i one zostały zaimplementowane w komputerowym narzędziu wspomagającym.

Należy zwrócić uwagę, że chociaż praca dotyczy zagadnień teoretycznych związanych z modelowaniem w dziedzinie inżynierii zabezpieczeń, to również posiada ona duży wymiar praktyczny, gdyż opracowane modele zostały zaimplementowane w postaci narzędzia do wspomagania pracy konstruktorów zabezpieczeń informatycznych, a także oceniających te zabezpieczenia. Zagadnienia implementacji przedstawiono skrótowo w rozdziale 12., odwołując się do dokumentacji technicznej narzędzia i jego wersji demonstracyjnej. Przedstawiono więc m.in. przykłady dotyczące: biblioteki projektowej, zawierającej setki uporządkowanych generyków i komponentów, kreatora projektów implementującego prezentowane w monografii diagramy czynności, wizualizacji wybranego łańcucha generyków i komponentów oraz automatycznego tworzenia zadania zabezpieczeń, a potem jego oceny.

Rozdział 13. stanowi podsumowanie (osiągnięte rezultaty, uzyskane doświadczenia, plany dotyczące rozwoju) pracy, przedstawionej w monografii oraz w innych publikacjach autora z nią związanych. Monografia zawiera załączniki dotyczące kolejno: podstawowej terminologii związanej z metodyką Wspólne kryteria, składni i semantyki OCL wykorzystywanych dla bardziej precyzyjnego opisu opracowanych modeli UML, zasad notacji wyrażeń kryptograficznych związanych z zapewnieniem zgodności z możliwą do pomocniczego zastosowania formalną metodyką UMLsec, objaśnienia konwencji stosowanych nazw oraz przykład dotyczący zadania zabezpieczeń dla systemu zaporowego.

INFORMATION FOR AUTHORS

The journal *STUDIA INFORMATICA* publishes both fundamental and applied Memoirs and Notes in the field of informatics. The Editors' aim is to provide an active forum for disseminating the original results of theoretical research and applications practice of informatics understood as a discipline focused on the investigations of laws that rule processes of coding, storing, processing, and transferring of information or data.

Papers are welcome from fields of informatics inclusive of, but not restricted to *Computer Science*, *Engineering*, and *Life and Physical Sciences*.

All manuscripts submitted for publication will be subject to critical review. Acceptability will be judged according to the paper's contribution to the art and science of informatics.

In the first instance, all text should be submitted as hardcopy, conventionally mailed, and for accepted paper accompanying with the electronically readable manuscript to:

Dr. Marcin SKOWRONEK
Institute of Informatics
Silesian University of Technology
ul. Akademicka 16
44-100 Gliwice, Poland
Tel.: +48 32 237-12-15
Fax: +48 32 237-27-33
e-mail: marcins@polsl.pl

MANUSCRIPT REQUIREMENTS

All manuscripts should be written in Polish or in English. Manuscript should be typed on one side paper only, and submitted in duplicate. The name and affiliation of each author should be followed by the title of the paper (as brief as possible). An abstract of not more than 50 words is required. The text should be logically divided under numbered headings and subheadings (up to four levels). Each table must have a title and should be cited in the text. Each figure should have a caption and have to be cited in the text. References should be cited with a number in square brackets that corresponds to a proper number in the reference list. The accuracy of the references is the author's responsibility. Abbreviations should be used sparingly and given in full at first mention (e.g. "Central Processing Unit (CPU)"). In case when the manuscript is provided in Polish (English) language, the summary and additional abstract (up to 300 words with reference to the equations, tables and figures) in English (Polish) should be added.

After the paper has been reviewed and accepted for publication, the author has to submit to the Editor a hardcopy and electronic version of the manuscript.

It is strongly recommended to submit the manuscript in a form downloadable from web site http://zti.iinf.polsl.gliwice.pl/makiety/.

To subscribe: *STUDIA INFORMATICA* (PL ISSN 0208-7286) is published by Silesian University of Technology Press (Wydawnictwo Politechniki Śląskiej) ul. Akademicka 5, 44-100 Gliwice, Poland, Tel./Fax +48 32 237-13-81. 2008 annual subscription rate: US$40. Single number price approx. US$8-15 according to the issue volume.