JACOBS
UNIVERSITY

Change Management in Common Criteria based IT Security Documentation

By

Milena Makaveeva

a thesis for conferral of a Master of Science in Computer Science

Prof. Dr. Michael Kohlhase
(Jacobs University)

Prof. Dr. Dieter Hutter
(DFKI)

Date of Submission: January 24th, 2010

School of Engineering and Science

Declaration

The research subsumed in this thesis has been conducted under the supervision of Prof. Dr. Michael Kohlhase from Jacobs University Bremen. All material presented in this Master Thesis is my own, unless specifically stated.

I, Milena Makaveeva, hereby declare, that, to the best of my knowledge, the research presented in this Master Thesis contains original and independent results, and it has not been submitted elsewhere for the conferral of a degree.

Milena Makaveeva
Bremen, January, 24, 2010

Acknowledgements

I would like to first of all thank my supervisor, Prof. Dr. Michael Kohlhase, for his continuous guidance throughout my entire research within the KWARC group and specifically for this thesis. His advice, motivation and encouragement made the completion of this thesis possible.

I would also like to thank my second supervisor, Prof. Dr. Dieter Hutter, for his guidance in the creation of this thesis and his support during my stay at the German Center for Artificial Intelligence in Saarbruecken.

I would also like to thank the members of the KWARC group for maintaining an exciting research environment, encouraging independent study and research.

Also, special thanks to my friends and university colleagues Norman Mueller, Anuj Sehgal, Georgi Chulkov and Teodor Cioaca for giving me advice or taking their time to proofread this document. I would especially like to thank Stefan Anca for his constant moral support, motivation, thesis proofreading and correction and for the template of this thesis.

Abstract

This thesis aims to alleviate the process of security evaluation based on the Common Criteria for Information Technology Security Evaluation (CC). It approaches the problem by integrating change management capabilities in the security certification process based on the Common Criteria.

The Common Criteria and its accompanying documents share a number of interrelations that render them unmaintainable by typical version control applications. Therefore, a semantics aware approach is necessary to solve the problem of their maintenance. In order to consistently propagate and integrate changes, we analyze the structure and relations within the documentation. On the basis of our observations, we established a semantic model for the CC that was later on translated to a model for the security documentation. These models serve to explicate the building structures of the documents and to determine proper paths for change propagation.

A software system, called CCWord, is proposed to support the creation and change management of IT security documentation. The system guides its user in the process of development of maintainable IT security documentation, enriched with structural semantics. Among the other functionalities directly or indirectly supported by CCWord are change detection, impact analysis and propagation. The system provides information about the steps necessary to restore the consistent state of the documentation, but the main decisions in the process are left to the user. Because of this interactivity, CCWord is considered to provide a semi-automatic solution to the problem at hand.

Contents

Chapter 1

Introduction

In the ever-competing business world, proving the security of a product is of great importance. The importance of information security has increased dramatically because of the move of open internal networks, to customers and business partners, the move towards electronic commerce and the increasing use of public networks like the Internet and intranets [HT06]. The widespread use of and dependency upon information and information processing require structured and organized protection of information. Actions of malicious users that place value on certain secure information, or of non-malicious users, that do not use the data in a proper manner, can often lead to unauthorized disclosure, modification or loss of use of secure information. This failure in confidentiality, integrity or availability can bring with itself serious negative consequences, ranging from significant recovery and restoration costs, costs of missed opportunities in cases when internal system functions are disrupted to a loss of company customer trust.

To gain necessary trust in an IT product or system, an independent evaluation of the security properties of the system needs to be performed.

The Common Criteria for IT Security Evaluation (CC) is an internationally recognized standard for IT Security Evaluation. [Com02a]. It is meant to be used as a basis for the documentation and evaluation of security properties of IT products.

The task set forth by this thesis is to provide change management capabilities in the process of IT Security Evaluation based on the Common Criteria. This is accomplished by enriching the IT Security documentation with formal semantics that is extracted as a result of the careful exploration of the Common Criteria and related to it documents and document collections. A way to detect the range and propagation scope of every single change is then suggested based on the provided semantics. The change integration itself is an interactive task left to the human factor. A software system, which is based on the conducted document analysis, is then provided to support the process of creation of semantically enriched security documentation and to integrate capabilities to deal with document changes.

The following sections describe the motivation for choosing this topic and provide a running example explaining how our research is going to be applied. Moreover, the thesis goals and research problems are also presented in this chapter.

1.1 Common Criteria Evaluation

The Common Criteria is an international standard for software security certification. Security critical products need to undergo security evaluation and pass it successfully in order to obtain a CC-based Certificate. CC Evaluation is performed from special evaluation entities over the so called Target of Evaluation (TOE) that consists of the IT product and its security documentation. The documentation is also known as evaluation evidence or developer documentation, as it is usually provided from the system developers.

The preparation of the evaluation evidence and other evaluation-related documentation can often require great amount of time and effort. By the time the work is completed, the product in evaluation may generally be obsolete. The point that produces overhead in the process of evaluation is the necessity to apply and integrate a change within the documentation. This overhead is due to the existing semantical interrelations between the different parts of the CC-based documents. A change in one part can, therefore, have widely spread consequences that need to be estimated before it can be integrated. To understand the process better we take a look into the evaluation life cycle.

1.1.1 Evaluation Life Cycle

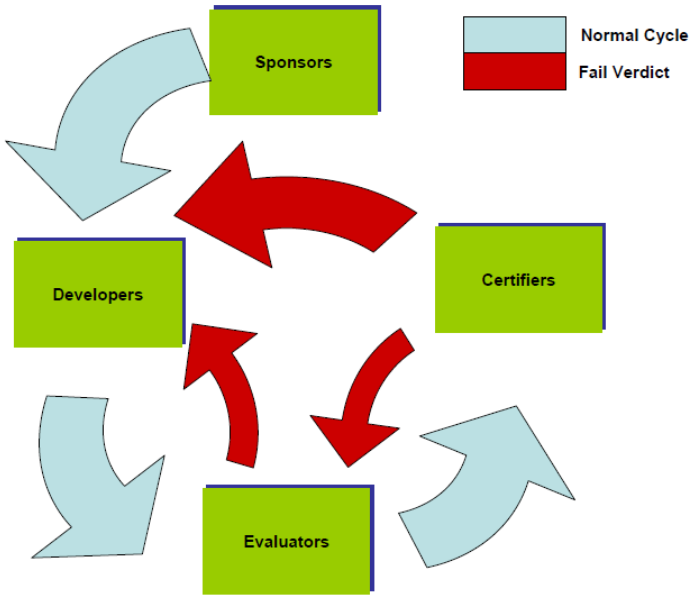


Figure 1.1: Evaluation Life Cycle

As shown in Figure 1.1, four different roles take active part in the process of evaluation. First comes the sponsor - the entity that requests the evaluation to be performed; this may be the customer or developer organization. Once a system is subject to evaluation, the

developers are supposed to generate the CC artifacts that will serve as the basis for evaluation. The evaluator is a specially accredited entity that the sponsor selects. The evaluators are those that assign a verdict of a "pass" or a "fail" to the security documentation. In cases of a pass verdict, in order to enhance the consistency of the evaluation findings, the final evaluation results are usually submitted to a certification process. Certifiers are the last participants in the evaluation life cycle. Certifiers review the formal evaluation results and approve or reject them. The certification process is the independent inspection of the evaluation results that leads to the production of the final certificate or approval, which is normally publicly available.

The process of evaluation can break at any point. The first loop is between developers and evaluators. In case the developer documentation is not approved by the evaluators the former have to revise their documents and even sometimes system and the evaluation process should be restarted. Once the documentation manages to get approved, it makes its way to certification, at this phase of the process, it is still at risk of being rejected and then sent back to any of the previous two phases.

Overall, we can already observe the cyclic nature of the process and anticipate the benefits we could obtain by detecting, propagating and integrating changes.

Even when having a guide, such as the CC at hand, the costs, effort and time of creating a proper documentation sometimes seem unaccounted for. In order ot speedup the evaluation, a possibility for management of change should be provided.

To provide for change management, the structure and semantics behind the CC documentation need to be understood and exploited. Once the structure of the documentation and how to make use of it have become clear, the developers of the documentation should make the structure explicit during the document creation. Manually annotating the semantics is a laborious task that often does not scale properly to the demands of the application. A system to create structured and semantically enriched documentation is therefore necessary. Such a system can not only help in laying grounds for management of change, but also will provide guidance for faster and easier document creation.

1.2 Running Example

In order to provide a practical motivation for this thesis, we consider a typical Common Criteria user scenario as demonstrated in the example below.

Consider the case of a database (named QuickQuery) that is supposed to provide a medium to low level security. A company ("Smart Solutions Ltd") [1] needs a reliable application to store and at the same time protect their data from unauthorized disclosure, modification and loss of use. They consider that QuickQuery may be suitable for their needs, but require a proof that this is actually the case. As a result, the personnel of "Smart Solutions Ltd" prepares a list with all security properties that are of interest to

[1] Disclaimer: All names in this thesis are fictional. Any similarities to real company or product names are pure coincidence and not our intent

them and then requests that QuickQuery undergoes an evaluation of the selected security properties with respect to the Common Criteria for IT Security.

The developers of QuickQuery are then supposed to in short term provide a proper, commensurate with the requirements of the Common Criteria documentation of all the security related parameters of the system. Ideally, most of the documentation would have been created parallel with the process of software development, but in the real world things may often be different. That's where, the role of the CCWord system proposed in this thesis emerges. The developer can use the CCWord as the tool that guides them through the process of document creation. CCWord makes it possible to create semantically enriched, maintainable, consistent and correct documentation

1.3 Thesis Contribution

This section discusses the research goals of this thesis and the approach undertaken to achieve them. The main objective of this project is to provide a solution to the problem of change management in IT Security Documentation pertaining to the Common Criteria. As a consequence of the conducted research, the thesis will answer the question presented in Table 1.1.

Can we speed up and alleviate the time-consuming and bulky process of security certification based on the Common Criteria for IT Security Evaluation by integrating management of change capabilities within the documentation?

Table 1.1: Research Question

Several subgoals emerge as immediate consequences of the main goal. To maintain the CC-related documentation with respect to change, we start with the objective to understand the effects of changes on the documentation. We are interested in how a change in one part of the document affects other document parts and what adjustments need to be performed to them as a result. This requires understanding of the structure of the documentation and its inter-relations. Therefore, our first goal is to determine the Common Criteria structure and derive formal semantics for the CC and the security documentation based on it.

We start with a detailed exploration of the Common Criteria and analyze the already existing CC structures and the possibilities to utilize the dependencies between them. Because we want to more precisely estimate the scope and effect of each change and the proper way to propagate it, we propose a way to structure the criteria into smaller units and to characterize their interrelations.

In order to determine formal semantics for the security documentation, we need to map the structural units and relations of the CC to units and relations in the evaluation evidence.

Our next goal is to understand when the system is in a consistent states and to formulate consistency constraints. Hereby the problem of management of change translates to the

problem of restoring a consistent state of the documentation after a change has been applied.

Then we explore the options to semantically enrich the evaluation evidence with the information about the derived through analysis structures and to exploit the integrated semantic information for its maintenance.

In order to ensure that the evaluator evidence is sustainable, we set forth to support its creation and maintenance by a tool aware of its underlying semantics. Our goal is to develop a software system that assists in the creation of semantically enriched, IT security documents. By providing CCWord, we aim to move the focus of document creation away from the structure and representation to the actual content of the documentation.

1.4 Structure

This thesis is organized as follows. Chapter 1 provides a general introduction to the problem of interest, presents the running example and formulates the thesis goals and the steps undertaken to provide a solution. The state of the art is presented in Chapter 3. Information over related research is provided for three topics of relevance to the problem at hand: management of change, requirements management and traceability and Common Criteria support tools. Chapter 2 introduces the reader to the Common Criteria and its supporting Common Evaluation Methodology (CEM) and their application in the process of evaluation. Chapter 4 analyzes and characterizes the existing Common Criteria structures and their interrelationships and proposes further substructuring. The types and properties of relations are considered in detail in Chapter 5. Chapter 6 serves as a link between the derived concepts and the proposed software system. It discusses the correspondence between the CC and the security documentation and deducts the properties of a system that would be able to create the different types of developers documents. Chapter refch:SoftArch introduces CCWord, the software system used to generate semantically enriched security documents and to support the management of change process. The last part of this document draws a conclusion to the work presented in this thesis and provides an outlook for future work and development that can be contributed to the project.

Chapter 2

Common Criteria Introduction

This chapter provides background information about the Common Criteria for IT Security Evaluation and introduces the other documents that take part in the process of CC-based IT Security Certification.

2.1 Introduction to the Common Criteria

The Common Criteria for Information Technology Security Evaluation (abbreviated as the Common Criteria or the CC) [wik, Com02a] is an international standard for computer security. The Common Criteria is based upon a framework in which computer system users can specify their security requirements, vendors can then implement them or make claims about the security attributes of their products, and evaluation laboratories can evaluate the products to determine if they actually meet the claims. The CC permits comparability between the results of independent security evaluations. [wik]. It does so by providing a common set of requirements for the security functionality of IT products and for assurance measures applied to these IT products during a security evaluation.

The CC addresses protection of information from unauthorized disclosure, modification, or loss of use. The categories of protection relating to these three types of failure of security are commonly called confidentiality, integrity, and availability, respectively. The CC may also be applicable to aspects of IT security outside of these three categories. Typical additional aspects to be considered are the authenticity of information and communication partners, as well as non-repudiation of origin or receipt of information.

In other words, the Common Criteria provides a technical basis and assurance that the processes of specification and evaluation of a computer security system have been conducted in a rigorous and standard manner. The CC is used as a warranty that the evaluation has led to a meaningful result that is mutually recognized between evaluation authorities. In order to further standardize the process, the evaluation of a system is done on the basis of one or more schema that give instructions on how to apply the CC. A common such schema is the Common Evaluation Methodology (CEM) [Com09]. A more detailed information of the structures of the CC and the CEM are presented in Section 2.2

and Section 2.5 respectively.

2.2 Structure of the Common Criteria

The Common Criteria document consists of three parts. Part 1 [Com02b] gives the introduction to the CC, defines the general concepts and principles of IT Security Evaluation and presents a general model of evaluation. Part 2 [Com02c] consists of components that serve as standard templates upon which functional requirements for IT products are based. They represents well understood security functional requirements that can be used to create trusted products reflecting the needs of the market. Part 3 [Com02d] establishes a set of assurance components that serve as standard templates upon which to base assurance requirements for IT Systems [Com02b].

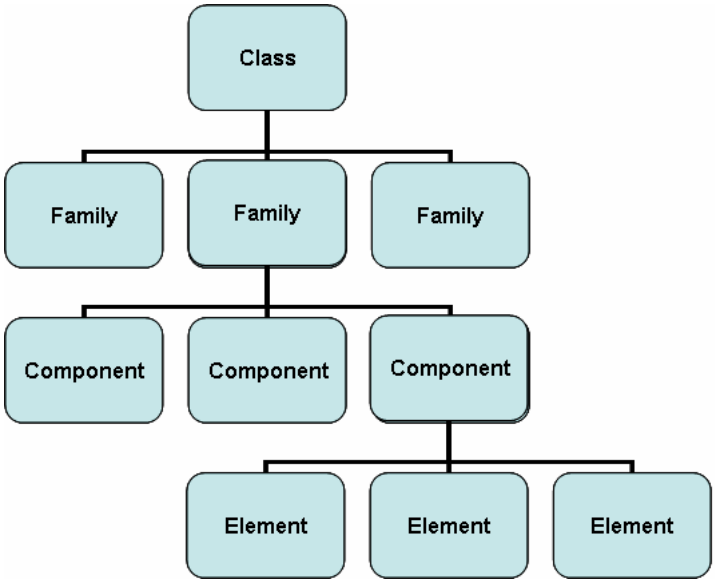


Figure 2.1: Hierarchy of Requirements

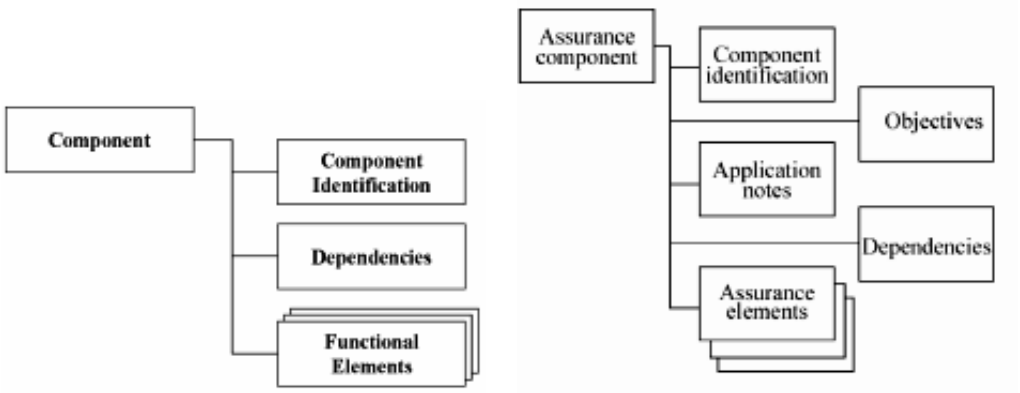
Part 2 and Part 3 share a common basic structure. They both take the form of catalogues that organize the security functional and security assurance requirements into families and classes. The organizational hierarchy of both the functional and assurance requirements is shown in Figure 2.1.

Classes are groups of components that share a common topic. They are structured into families. As an example the assurance class ADV (Development) contains the assurance families "ADV_FSP - Functional Specification" , "ADV_TDS - TOE design", and other families.

Families are groupings of components that share a more specific focus but may differ in emphasis and rigor. The members of the family are termed components.

Components are the smallest selectable units and are usually used in a Security Target to formalize the functional requirements (Part 2 components) or to specify the rigor

and depth of evaluation (Part 3 components). They may be partially ordered to represent related hierarchical sets. Hierarchical here means that ADV_FSP.2 extends the requirements of ADV_FSP.1 and so on. The structures of the security functional components from CCPart 2 and of the security functional components from CC Part 3 in this order are represented in Table 2.1 [Com02c, Com02d].



Security Functional Compopnent Security Assurance Component

Table 2.1: Component Structures

Elements are the lowest level expressions of security needs that are verified by the evaluation. They comprise the security assurance components of the CC. The security assurance elements are additionally classified as developer action elements, content and presentation of evidence elements or evaluator action elements. They serve as guidelines for developers and evaluators respectively.

Part 3 is also based on Evaluation Assurance Levels (EAL) . They are packages consisting of assurance requirements, drawn from CC Part 3. Each evaluation assurance level provides a certain level of security. Seven hierarchically ordered evaluation assurance levels are predefined in the CC. The ordering is inasmuch as each Evaluation Assurance Level represents more assurance than any lower level. Increase in assurance is usually accomplished by substitution of a hierarchically higher assurance component from the same assurance family, or by addition of assurance components from other assurance families. The Evaluation Assurance Level is selected for an IT System prior to the process of evaluation. It determines a standard set of security requirements that the system needs to comply with.

2.3 Topic of Evaluation

Security, in the context of the Common Criteria, is concerned with the protection of assets. Assets are entities that someone places value upon, such as contents of files on servers, election results, availability of an electronic commerce process, but as the value is highly subjective almost anything can be an asset. The environment where the assets are located is called Operational Environment. The environment can be physical as well as virtual.

Many assets are in the form of information that is stored, processed and transmitted by IT products to meet requirements laid down by owners of the information. Information owners may require that availability, dissemination and modification of any such information is strictly controlled and that the assets are protected from threats by countermeasures.

Safeguarding assets of interest is the responsibility of owners who place value on those assets. Actual or presumed threat agents may also place value on the assets and seek to abuse assets in a manner contrary to the interests of the owner. Examples of threat agents include hackers, malicious users, non-malicious users who sometimes make errors and accidents. These threats lead to the necessity of imposing countermeasures. These countermeasures may consist of IT countermeasures, such as firewalls and smart cards and non-IT countermeasures, such as guards and safety procedures. Owners of assets may be held responsible for those assets and therefore should be able to defend the decision to accept the risks of exposing the assets to the threats. Defending this decision is the goal of the evaluation process. **Evaluation aims to demonstrate and prove that, the countermeasures are sufficient, i.e. all threats to the assets are countered, and that the countermeasures are correct, i.e. they do what they claim to do.**

2.4 Security Documentation

The security documentation or also called evaluation evidence specifies the security properties of an IT product or system. Because, it is usually created by the system developers, it is also referred to as developers documentation within this text. The security documentation consists of one or several Protection Profiles, a Security Target (ST) and a Target of Evaluation(TOE).

2.4.1 Security Target and Protection Profiles

Security Target(ST) Formally defined, the Security Target(ST) is an implementation-dependent statement of security needs for a specific TOE. The Security Target is usually based on one or several Protection Profiles.

The ST states the security problem of the software system and a possible solution to it. Figure 2.2 represents the core content of the Security Target document. The security problem of the system is comprised of threats, Organizational Security Policies(OSPs) and secure usage assumptions. Solution is provided through the security objectives in the form of two sub-solutions: security objectives for the TOE - implemented by security requirements allocated to the TOE and security objectives for the operational environment - implemented by security requirements allocated to the systems that interact with the TOE. Additionally, a security objectives rationale is provided, showing that if all security objectives are achieved, the security problem is solved: all threats are countered, all OSPs are enforced, and all assumptions are upheld. The security objectives and the solution to the security problem are then formalized by security functional and assurance requirements derived from CC Part 2 and CC Part 3 respectively. All relevant to the developer

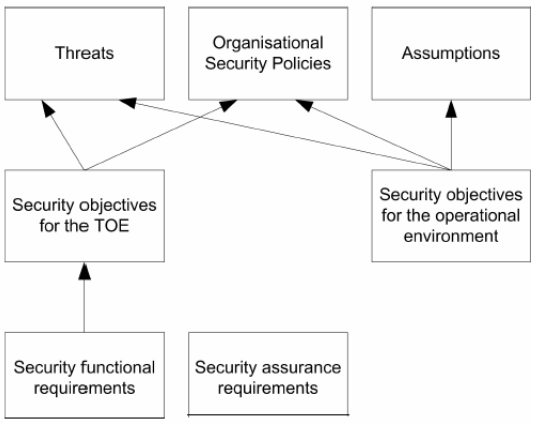


Figure 2.2: Security Target and Protection Profile Objects

documentation CC components as well as the Evaluation Assurance Level are defined in the Security Target.

Protection Profile(PP) A Protection Profile(PP) is an implementation-independent statement of security needs for a TOE. It describes the security problem for a family of IT products. Each Protection Profile specifies requirements for a family of IT products with common characteristics and security needs and can also inherit requirements from other Protection Profiles.

Relationship between the Protection Profile and the Security Target Both the Security Target and the Protection Profile identify security problems. The PP identifies the security problem for a family of IT products, while the ST is targeted at a particular IT product. A Security Target may conform to one or several Protection Profiles. That would mean that the security problem identified by the Security Target is a combination of the problems identified by the Protection Profiles plus some additional system specific requirements.

2.4.2 Target of Evaluation

The Target of Evaluation(TOE), whose written part is also known as assurance documentation, is composed of a set of software, firmware or hardware, accompanied by a guidance. When we refer to the Target of Evaluation within this text, we mean its pertaining assurance documentation.

Relationship between the ST and the TOE Documentation The purpose of the Security Target is to state the Security Problem and then show that the countermeasures

undertaken are sufficient to handle all appearing threats. The purpose of assurance documentation is to ensure that the security measures as described in the Security Target are applied properly. The fact that the assurance documentation shows that exactly the countermeasures described in the Security Target are correct means that throughout the whole TOE security documentation a number of references towards different fragments of the Security Target are made. For example, if during the exploitation of an IT Product a new threat appears, the Security Target needs to be updated with a description of the threat and the respective countermeasures applicable to it. This in turns means, that a proof to the correctness of the countermeasures should be provided in the assurance documentation of the TOE.

2.5 Common Evaluation Methodology

The Common Evaluation Methodology(CEM) [Com09] is a document companion to the CC. It focuses on actions the evaluator must take in order to determine whether or not the CC requirements have been complied with. The CEM defines the minimum set of actions to be performed by an evaluator in order to conduct a CC evaluation using the criteria and evaluation evidence defined in the CC.

2.5.1 Security Evaluation

In the context of Common Criteria, evaluation is a judgment about the conformance of an IT Product to its security requirements. The goal of evaluation is to demonstrate and prove that the provided information related to the system security is sufficient and correct. The subjects of evaluation are the Security Target, any available Protection Profiles it conforms with and the Target of Evaluation (TOE) as defined in Section 2.4.

The evaluation of the Protection Profiles will be left out of our discussion, as we assume the the developers are going to use Protection Profiles that already meet the CC requirements.

In addition to the PP evaluation, there are two other evaluation processes. In the first one, the Security Target of the IT product is evaluated, that is done to determine that the countermeasures undertaken are sufficient, i.e. all identified threats to the assets are countered. In the second process the TOE is being evaluated against the evaluated ST and a corresponding Evaluation Assurance Level, defined in ST, in order to obtain grounds for confidence that the claimed security objectives are correctly achieved. While the former evaluation seems to an extent straightforward, as it asks for the consistency of a single document, the latter one is based on the former one and usually involves dealing with bulky documentation, hence becoming much more complex.

2.5.2 Evaluation Process

The principle input in evaluating a TOE is the so called evaluation evidence. It includes the TOE itself and the Security Target, but will usually also include input from the development environment, such as design documents or developer test results. The output is a verdict of pass, fail or inconclusive, determining the outcome of the evaluation.

The evaluation of an IT system consists of applying the security requirements to the evaluation evidence. As explained earlier, all security assurance components relevant to the process of evaluation are defined in the Security Target of the system. For the purposes of evaluation, the evaluator needs to perform the respective CEM subactivity to each component.

The evaluation assigns a verdict of pass, inconclusive or fail to the requirements of the CC and not to those of the CEM. The smallest structure to which a verdict is assigned is the evaluator action element. A verdict is assigned to an applicable CC evaluator action element as a result of performing the corresponding CEM actions and its constituent work units. Work units are the most granular level of evaluation work. Each CEM action comprises one or more work units, which are grouped within the CEM action by CC content and presentation of evidence or developer action element. The work units are presented in the CEM in the same order as the CC elements from which they are derived.

The possible CEM verdicts are pass, fail or inconclusive. A pass is defined as an evaluator completion of the CC evaluator action element and determination that the requirements for the PP, ST and TOE under evaluation are met. Conditions for a fail verdict is defined as an evaluator completion of the CC evaluator action element and determination that the requirements for the PP, ST and TOE under evaluation are not met, or the evidence is incoherent or obviously inconsistent. All verdicts are initially considered inconclusive and remain so until either a pass or a fail verdict is assigned.

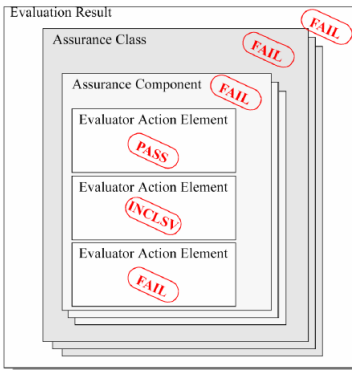


Figure 2.3: Verdict Assignment Rule

As shown in Figure 2.3 [Com09], it is enough for a single element within the component to fail, i.e. a single work unit to evaluate to fail, for the whole component, family and class to be assigned a fail verdict and hence the overall verdict to be a fail.

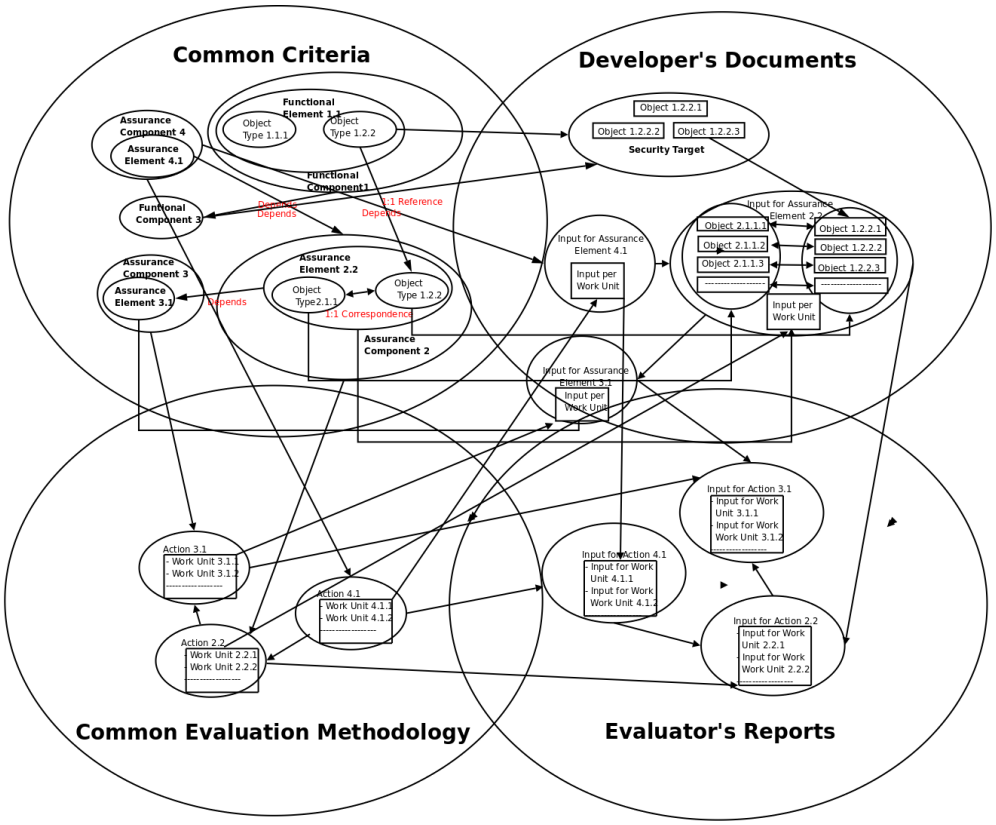


Figure 2.4: Dependencies - Global Scale

2.5.3 Evaluator Documentation

The Evaluator documentation consists of an Evaluation Technical Report and Observation Reports, where the former presents the overall verdict and its justification and the latter represent reports that request clarification or identify a problem during the evaluation. Either a justification or an identification of a problem, the evaluator work is based at the same time on the Common Criteria, the Common Evaluation Methodology and the evaluation evidence. In the next section, we explore the relation between the different types of documentation.

2.6 Document Relations in the CC-based Evaluation

Figure 2.4 although seemingly overloaded, represents only a small subset of the multitude of dependency relations that exist for the collection of CC-related documents that participate in the security evaluation process. These relations are explained in the sections to follow.

2.6.1 Relations between the CC and the CEM

The two documents that serve as a basis for evaluation CC and CEM share a straightforward correspondence. Figure 2.5 [Com09] gives a general overview of how CC and CEM structures relate. Every class, component, evaluator action element or developer action element from the CC Part 3 corresponds to a respective activity, sub-activity, action or a group of work units from the CEM.

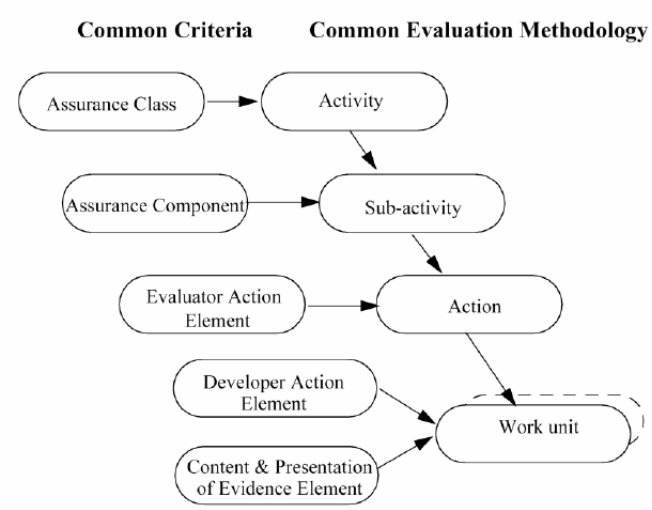


Figure 2.5: Mapping of the CC and CEM Structures

2.6.2 Relations between the CEM and the Security Documentation

The dependence of the CEM on the CC and the fact that the Developer documentation should comply with the requirements of evaluation lead to a backward dependency of the developer documentation on the Common Evaluation Methodology. The work units described in the CEM can be considered a guide for the developer as well. A simple rule of how to read this guide would be to replace sentence parts approximately synonymous to *"The evaluator should verify..."* with sentence parts approximately synonymous to *"The developer should ensure..."*

2.6.3 Relations between the Security Documentation and the Evaluator Documentation

In Section 2.6.1 and Section 2.5.3, we gave a short overview of how both the Developer and Evaluator Documentation base their contents on the CC and the work units of the CEM.

Here we present the direct relationship that exists between the Developer and Evaluator documentation as a result.

The security assurance documentation provides evidence that the CC functional requirements selected in the Security Target are correctly implemented. As a result, it contains information for every single content and presentation of evidence element of the defined in the ST assurance components.

Either a justification or an identification of a problem, the evaluator work is mostly based on the components and their substructures, or on the fragments of the Security Target if it is a ST documentation. Transitively, we have identified a one-to-one correspondence between developer documentation and evaluator documentation.

To simplify the system we propose, we abstract from the influence of CEM on the writing of Developer Documentation and concentrate our analysis on the correspondences between the Common Criteria and Developer Documentation only.

Chapter 3

State of the art and Related Work

The research question of this thesis can be viewed as a combination of two different perspectives, namely Management of Change and support for CC documents development. In this chapter we present the state of the art in both areas so far and draw some relations to our problem at hand.

3.1 Management of Change

Management of change has been a popular topic of research in recent years. Various solutions to the problem of maintaining changing data are already available - some of them tailored to a particular area, while others aiming to provide a general solution. In this section, we discuss several of the most popular approaches existing.

Currently, Document Management Systems are designed to coordinate the collaborative creation and maintenance process of documents through the provision of a centralized repository. These Management Systems focus primarily on management over the structure and content of the document. Relations between and within documents as well as the effect of changes on these relations are largely neglected, although information reuse and distribution could seriously benefit from such a relation management. And even the few systems that actually take the dependencies between and within documents into consideration usually fail to achieve that with a high enough level of precision.

3.1.1 Version Control

One of the classical examples of management of change applications are the version control systems [Tic85], which track and save the history of file changes over time. These systems are useful when several developers work concurrently on overlapping data over a long period of time.

With version control systems local copies of remote files can be obtained for reading or editing. After modifications are performed locally, version control provides means to integrate your changes with the remote version of the files. File modifications are saved

to the server in a way, that ensures no data is concurrently modified. One can also keep a history of files and version number associated to each code release, while managing a concurrent access to the source files maintained by the server. Version control systems can also be used to restore to previous version of a given file or to merge several existing files into one.

The benefits of version control systems are manifold. A good version control system helps to protect the integrity of data. By keeping a revision history, there is no worry that if code is removed in an edit on one day, it will be lost when it is determined a week later to have in fact been necessary.

Version control systems compute the differences between versions based on a simple line by line comparison using the standard Unix *diff* program. According to [CRGmW96], such systems fail to provide adequate version control for structured data, because they do not understand the hierarchical structure information contained in such data sets. As an example, in formats such as XML, where tags are placed on data segments to indicate context, syntactically different documents may turn to be semantically equivalent.

The version control systems are therefore able to provide only a limited management of change capabilities for the CC-based IT Security documentation. More specifically, they will fail to interpret the semantic interrelations between the constituent document parts. Therefore, version control management of change can leave the documentation in an inconsistent state.

3.1.2 Semantic Management of Heterogeneous Documents

Existing change management tools are either domain independent supporting only simple syntactical relations or they are tailored(and thus restricted) to a specific application domain because they make use of document semantics. [Hut09] and [Mül07] aim at generic, semantics-aware, change management systems, that can be applied in various domains of interest, by instantiating generic mechanisms to individual document types.

Locutor [Mül07] is a system used to provide ontology-driven management of change built on informal document engineering processes. Documents are regarded as structured compositions of information units and the dependencies between fragments of documents are considered. Locutor also uses the notion of separation of documents into content and narrative layer, whereas the presentational order of information units is represented on the narrative layer while the information units themselves and the ontological relations between them are placed in the content layer. Locutor is mainly used as a tool where the users register their documents, according to the proper ontological classification of their parts and informs the system about the dependencies over the documents. If a document part is changed, all the users that have document dependencies on the part get informed. With all that Locutor takes an important step towards semantically based management of change.

[Hut09] defines change management over heterogeneous (XML-) documents. Change management is viewed as decomposable into the notions of mechanism and semantics, while the former can be fixed, the latter depends on the type of document collection.

The semantics is encoded in document-type specific and collection-type specific ontologies formalizing the dependencies in individual document types as well as the interrelations between them.

The process starts with documents represented as XML trees that are later turned into document graphs, as a result of enrichment with semantic annotations. These annotations become additional links or nodes in the graphs that make relationships and dependencies explicit. They reflect semantics or derived knowledge about the documents and are used to define sophisticated semantic constraints on document collections. Consistency is identified as the satisfaction of a set of consistency constraints that are formulated as predicates operating on document graphs. From a technical point of view consistency are just relations between the nodes of the document graph. The paper analyses the consequences of changes made in one document to others, and engineers the synchronization steps necessary to obtain a consistent document collection.

Both [Hut09] and [Mül07] generalize the approach undertaken by many consistency-preserving, change management systems. They proposes change management capabilities on top of a documentation enriched with structural semantics. Another popular approach is requirement tracing as represented in Section 3.1.3. A part of this approach is placing traces throughout the document of interest and following their paths in case a change needs to be propagated and integrated.

3.1.3 Requirement Management Systems

A prominent examples of document maintenance is the area of requirement change management. During the process of Software development various specifications need to be created. Making a small change in one document can cause a number of adjustments in various other development documents.

Requirements are properties to which a software should confirm. Requirement change management is of great interest to software engineering due to the general volatile nature of software requirements that often originates from an incomplete knowledge about the domain of interest. It is concerned with the capturing of project requirements, tracing requirements through project artifacts, measuring and identifying the impacts of changing requirements within project and answering whether all requirements of a given project have been met [MCV04] [tel]. Requirements management processes span the entire v-model development lifecycle - from inception when requirements are gathered and defined, to the end of development when final testing is carried out with respect to the initial requirements.

Requirement change management is mostly enabled through requirement tracing [RJ99]. Requirement traceability is concerned with the question of how to acquire and maintain a trace of requirements along their manifestations in development artifacts at different level of abstraction. This is a key technique for software maintenance because the information about relationships between different artifacts that is provided by these traces allows to access the impact and scope of planned changes.

Telelogic Doors Telelogic Doors [tel01] is a system for specification and maintenance of software requirements based on requirement traceability. It is designed to solve the problem of software requirements maintainability for complex systems and software. It provides capabilities to easily store requirements, link them with project artifacts and handle applied changes in automated way. In the basis of operation of DOORS is linking sets of information to create relationships between them and establish an audit trial of how one set of information was derived from the other. An automatic change notification feature alerts the user to any change in the linked information, so that an assessment can be made of the impact of the change. On the basis of DOORs, developers can determine if there exists enough evidence that the user requirements defined in the software specification are covered and achieved by the software.

An analogy between the security requirements as defined in the CC documentation and the software development requirements can be drawn. For both types of requirements, we would like to follow the relationships and changes of a requirement from the moment of its inclusion in the documentation until the moment the documentation is in an unchanging state. For both requirement types, we would also like to relate every requirement to the information that proves its satisfiability. Doors is also of interest to us because it represents a complete system, in which artifacts are both created and maintained. In order to provide change management capabilities to the Common Criteria, we follow a similar pattern. Our system supports both the creation and maintainability of security requirements and ensures that sufficient information is provided to guarantee consistency.

Requirement change management in Healthcare

3.2 Common Criteria Tools

A lot of research exists on how to ease the development of CC-based IT Security Documentation. Many systems supporting the processing of security documents are already available on the market, all of them serving the same goal - to facilitate the widespread acceptance of use of CC. Most available solutions, however, are either partial, or support only the creation but not the maintenance of the documentation. In this section, we discuss the most popular available Common Criteria tools.

3.2.1 CCToolbox and CC Profiling Knowledge Base

The CC Toolbox provides a interface to the Common Criteria for Information Security Technology Evaluation [CCT01]. It is an application that helps Protection Profile (PP) authors and Security Target (ST) authors in drafting PPs and STs.

CCToolbox is especially helpful for novice users, as it provides the ability to review different CC structures.

Since the paradigms of creating a ST and PP are similar, the user interface of the CCToolbox is used to accommodate both sets of tasks. CCToolbox consists of two tools

- one of them for creating Protection Profiles and the other for Security Targets. These tools share a common interface. A radio button is used to switch between the two tools. Users of the CC Toolbox can maneuver through the tool, specifying functional or assurance requirements, Security Objectives, Threats, Policies, etc., and defer the decision of which type of report they wish to produce until a later time in the process. The output of the CC Toolbox is a skeleton report that can be polished into a PP or ST depending on the user's choice.

The CC Profiling Knowledge Base is a database of sample security engineering information developed to support the CC Toolbox. The Knowledge Base contains sample policy, threat, and assumption statements based upon Department of Defense needs. In addition, it contains security objectives and recommended CC components with rationale and mappings tying together the TOE security environment, the security objectives, and the CC components. The CC Profiling Knowledge Base is extensible and communities of users can modify and add to the Knowledge Base to accommodate their common needs.

The CC Toolbox and CC Profiling Knowledge Base provide support for the creation of Security Targets and Protection Profiles. We could not find any information so far that these tools will provide support for creation of assurance documents or that the created with them documentation will be informed about the Common Criteria internal semantics.s

The CC Toolbox and CC Profiling Knowledge Base assist the creation of the founding IT Security documentation and could have therefore been used as tools to extend into a system that solves the CC change management problem. However, these tools have the disadvantage that their most recent versions date back to 2001. The tools have not been developed since then and their source code is no longer publicly available at the website of the NIAP(National Information Assurance Partnership of US).

3.2.2 CC Ontology Tool and the Common Criteria Design Toolbox

Most of the CC processing tools and representations rely on the specification of a CC ontology.

The CC Ontology tool provides an ontology mapping of Common Criteria's Assurance Requirements. [EFGW07]It covers the entire domain of CC and provides a means to query the data structures by using SPARQL. Based on this ontology a tool to support the CC certification process was created. The tool lists all CC Classes including their families and concepts, also filters them corresponding to EAL. It also provides for linking of relevant documents with the specific components.

[Nyg07] provides a tool for writing security specifications. Its main goal is to help developers in creating Protection Profiles and Security Targets for their products as well as to ease the book-keeping process while presenting users with a number of relevant choices and to further provide the option to validate the created documentation. The Common Criteria Design Toolbox is developed in the Microsoft .NET Framework Version 2.0. Furthermore parts of the Toolbox are using the dictionary functionality of WordNet

2.1. The tool obtains information for the CC structures by parsing a pre-created XML format of the Common Criteria.

Documentation created with the help of the tool is stored in the widely known and commonly used XML format. Using a standardized format makes the created documentation available to a wider audience. One drawback of the design toolbox is its limited scope of operation. It provides for the development of only Protection Profiles and Security Targets and does not propose any methods or lay any grounds for further development.

Overall the idea behind the CC Ontology tool and the Common Criteria Design Toolbox share a lot of similarities with our idea of supporting automated security documentation. Similarly to the former, we consider the semantic model of the CC and the security documents collections. Similarly to the latter, we convert the documents into an XML format that is widely understood and easily parsable. The main differences lay in the coverage of the documentation provided by the tools and the maintainability options they provide.

Chapter 4

Common Criteria Interpretation

In this thesis, we tackle the problem of maintenance of document collections based on the Common Criteria for IT Security Evaluation. In order to provide management of change capabilities, it is necessary to predict the effect of every single change. We are particularly interested in the question how a change propagates within the documentation. In the context of this work, change propagation for a change applied to an entity A, means the determination of all entities B that are either directly dependent on A, or there exists an entity C, such that a change can be propagated from A to C and B is dependent on C. If we consider the entities to be nodes in a graph and the dependencies to be edges, then a change in a node A will be propagated to all nodes B, for which there exists a path in the graph that starts with A and ends in B. By this definition change propagation is a property that directly depends on the document graph structure and not on the type of modifications.

We need to find semantic models that turn the CC-based documents into document graphs. Artificially created demonstrational examples as well as practical real-world samples of CC-related developers and evaluators documentation were analyzed in order to obtain domain specific knowledge and to determine the underlying CC structural semantics. We identify CC structural units that serve as nodes in the document graph and CC relations between them that correspond to the connecting edges. Based on this information and the mapping between the Common Criteria and Security Documentation, we derive a semantic model for the security documentation and the ability to propagate changes in it.

This chapter introduces the correspondence between the Common Criteria and the Security Documentation, represents a possible semantic model for the CC and then provides a mapping from it to a semantic model for the security documentation.

4.1 Correspondence between the Common Criteria and the Evaluation Evidence

The Developer documentation is also directly based on the CC. Figure 4.1 visualizes the

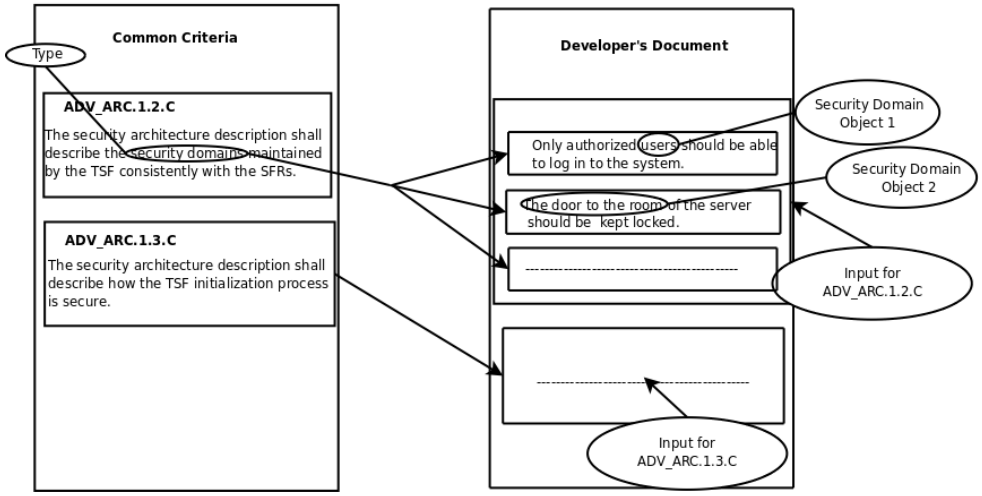


Figure 4.1: Relation between CC and Developer's Documentation

correspondence between the Common Criteria and a security document based on it. To the left of Figure 4.1 is the CC document in which structural elements, their characteristics and their dependencies are identified. To the right is an IT security document based on the CC. The developer documentation can be considered as a collection of information units, each of which corresponding to some CC structural entities. Similarly to [Mül06], we consider "information units" to be "tangible/visual text fragments potentially adequate for reuse that constitute the content of documents". The developer documentation contains information for only the structural units derived from assurance components of the Common Criteria Part 3.

The relations between the Common Criteria structures therefore directly transfer to relations between the corresponding information units in the security documentation. Input in the security documentation, will be provided as per either a Content and Presentation of Evidence Element or per any CC substructure that is of higher granularity. For every assurance element of the Common Criteria, respective textual fragments will exist in the TOE documentation. In most cases, apart from text, these fragments need to include input for the subunits contained in the CC element at hand. The subunits identified in the CC are presented in Section 4.2 below.

4.2 Common Criteria Semantics

The Common Criteria can be analytically decomposed into structural units and relations between them. The structural units consist of classes, families, components, elements, object types and object type references. The first four of them are notions that exist in the Common Criteria and are already introduced in Section 2.2, while the last three are defined by us. All structural units can participate in relations to other structural units. In

this document, we distinguish between dependencies and structural relations. Further in this section, we introduce the formal semantical model for the Common Criteria proposed by us.

Dependencies and Dependency Decomposition Dependencies are relations defining that a CC structural unit A depends on a CC structural unit B, i.e. any changes in B are likely to affect A. Dependencies provide links along which changes will be propagated. The CC already explicitly denotes some dependencies between components. The description of each component in both CC Part 2 and CC Part 3 contains a list of the components that it depends upon or is hierarchical to. These dependencies are useful when writing Protection Profiles or Security Targets. For example, whenever some functional or assurance component A is added to the aforementioned documents all components that A depends on should also be added in order to preserve consistency. However, they are too general for the purpose of maintaining the whole security documentation. Usually, the textual content that corresponds to a single CC component amounts to several pages. Knowing that a change has occurred or needs to be implemented within such range is not going to provide a considerable improvement in change management. Clearly, it is necessary to decompose dependendencies to dependencies between more granular information units. The elements are the smallest structural units identified in the Common Criteria. One possible solution is to replace the dependencies between components with dependencies between the elements of these components. Decomposing component-to-component dependencies to dependencies between elements is always possible and will be applied to all components from the Common Criteria Part 3.

We aim at the maximum possible decomposition of dependencies. Therefore we would like to, wherever possible, translate the dependencies between elements to relation between elements substructures.

Identifying substructures and structural relations The CC does not identify any more granular structural entities than elements. By careful exploration of the Common Criteria, we noticed that the CC elements often ask for identifying groups of entities with common properties, such as threats, countermeasures, security objectives, security interfaces, etc. Maybe, they can provide for a more granular substructure than elements. We called the notion of such groups *object types*. Another observation we made is that often inside the Common Criteria new *object types* are not introduced, but references to existing ones are made. We call this references to object types *object type references*.

We are interested in the interrelation between *object types*. They are established inside the Common Criteria through binary relations between *object type references* which we call *structural relations*. Therefore *structural relations* can be considered indirect relations between *object types* as shown in Figure 4.2. The direction of the arrows in Figure 4.2 shows the direction of the dependencies between structures. The *structural relations* depend on the *object types references*, which in turn depend on the *object types*.

To understand the notions better, lets take a look at the example presented in Table 4.1.

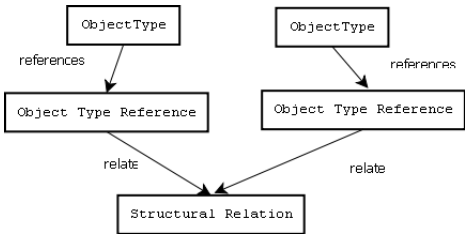


Figure 4.2: A CC Structural Relation

In the example, the object types of the Security Domains of the Target of Evaluation

ADV_ARC.1.2C *The security architecture description shall describe the security domains maintained by the TSF consistently with the SFRs.*

Table 4.1: Object Type Reference and Structural Relation Example

Security Functionality(TSF) and of the Security Functional Requirements are references in ADV_ARC.1.2C. The object types themselves are defined for the first time in another CC element. We can notice that a single element identifies two object type references and then defines a structural relation between them.

By decomposing dependencies between components, we derive dependencies between elements. **In order to ensure that the scope of a change is caught with the highest possible precision, changes will always be transferred over the dependency links of the smallest CC structural unit containing the change.** Whenever a possibility for that exists, dependencies between the CC elements will be decomposed to dependencies between object types, object type references and structural relations defined in this elements. These dependencies will look as shown in Figure 4.2.

Note that the CC structural units defined by us are present in the elements of the common criteria in a natural language format. For the purpose of this thesis, decomposition of dependencies, analysis of the CC elements and extraction of relevant CC structural units and relations is performed manually and relies on the judgment of a human factor.

4.3 Mapping between the CC and the Security Documentation

The semantic model of the Common Criteria is introduced in Section **??**. As it was mentioned before, it can be translated to a formal semantics model for the security documentation based on the dependency links between the Common Criteria and the security documentation.

In this section we describe how the structural units of the Common Criteria are mapped to the structural units of the security documentation. The data types of the Common Criteria and their corresponding counterparts in the security documentation are shown in Table 4.2. For translation, we consider the structure of the CC Part 3, as it is the structure

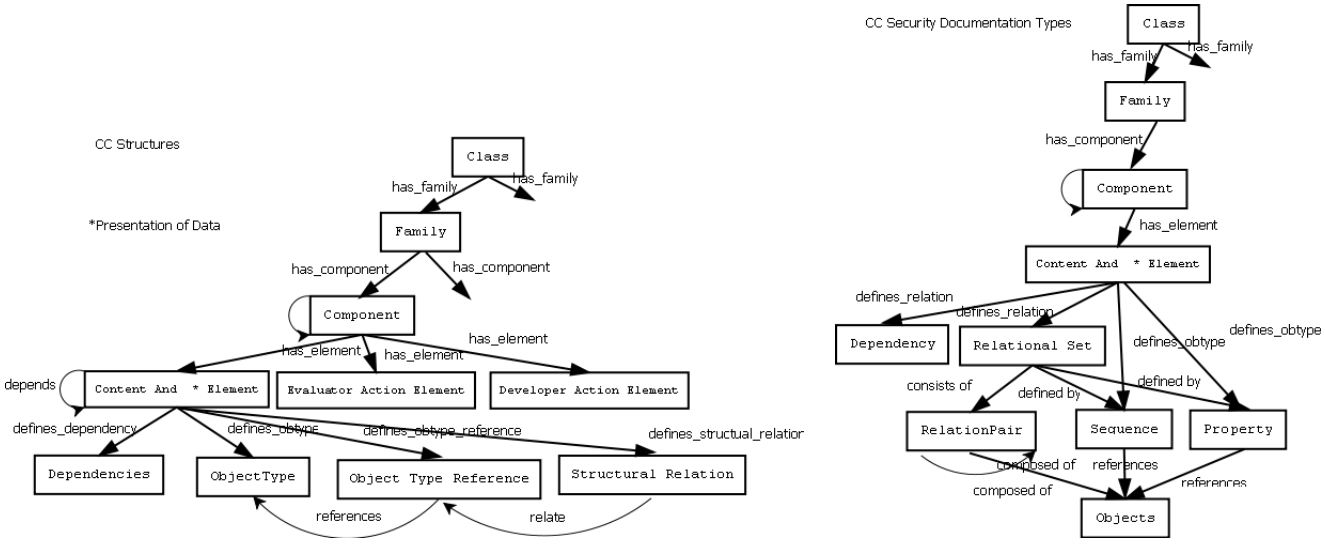


Table 4.2: Data Structures

to which the developer documentation directly corresponds.

We distinguish the following structural entities of the security documentation: classes, families, components, elements, objects of a type, sequences, properties, and relation pairs. Some of them directly correspond to structural units from the Common Criteria while others such as sequence serve mainly as auxiliary data units necessary to store information. To establish the relations of the entities we distinguish between declarations and definitions of dependencies and structural units .

Both the CC and the security documentation are composed of classes and families. Classes and families have mainly organizational and representational functions. They are used to provide for the organization and easy navigation between the structural units of the Common Criteria and the security documentation. Components are also the same for both types of documentation. Elements are mapped to other elements, where their type is preserved.

CC Elements are where the structure of the Common Criteria and that of the security documentation start to differ. One difference is that only the Content and Presentation of Evidence Elements are present in the developers documentation. That is because only those elements have corresponding textual content there. Content and Presentation of Evidence Elements, which we for short will call CC elements or just elements from here on, are the most complicated data structures due to the other, more granular structural entities they define.

All structural units of the CC are mapped to a declaration and a definition of these units in the developers documentation.

The notion of an object type exists in the security documentation as well. Object types are instantiated in the evaluation evidence as what we define as groups of *objects* of the type. They specify entities of the types. For example, *threats* is an object type, while *malicious users* and *malicious entities* are objects of this type. Objects are the smallest structural units the meaning of which is preserved independent of context. Each object has a type property and a unique name within the type and is associated with a text range in the security documentation.

When we talk about objects, we are in most cases interested in the notion of object groups rather than in individual objects. For example, the CC elements often require an enumeration of all objects of a certain kind. All objects of a type can be viewed as the biggest existing object group of the type. We are, however, usually not interested in all objects of a certain type but only in those that should participate in a certain relation, or that should be defined in the developers documentation due to the requirements of a CC Element.

Sequences and Properties represent groups of objects of a certain type. They correspond to the object type references of the Common Criteria and are instantiated as a set of references to objects from the defined type. The sequences are a selected set of CC objects of a certain type that consists of a subset of all defined in the documentation objects of this type.

Each relation identified in the Common Criteria is mapped to a declaration and a definition of the corresponding relation in the Security Documentation. The declaration tells the security documentation that such a relation is supposed to be part of it and how it has to be defined. The definition assigns structural units from the developer documentation as participants in the relation.

Dependencies between CC structures are mapped to dependencies between their corresponding security documentation structures. The declarations of dependencies state that entity A depends on an entity B. In the security documentation they are represented as dependency links between already defined structures.

Structural relations also have a corresponding declaration and definition in the security documentation. Structural relations are declared in the security documentation by a pair of sequences, properties or other relations in accordance with the formulations in Chapter 5. In the security documentation, structural relations are instantiated by sets of relation pairs. Relation pairs consist of references to two entities, each of which either an object or another relation pair. Similarly to objects, relation pairs are assigned a text range in the security documentation. Logically, the text range of the relation pair is always dependent on the text ranges corresponding to its constituent entities.

4.4 Classification of elements

As mention in Section 2.2, the CC components consist of three types of elements-developer action elements, content and presentation of evidence elements and evaluator action elements.

The developer action elements and the evaluation action elements of the Common Criteria are used as a guidance to the process of security evaluation. The CC content and presentation of evidence elements are those that establish requirements over the security documentation and represent the direct link between the Common Criteria and the security evaluation, as textual content needs to be provided in the developer documentation per content and presentation of evidence element.

Therefore we propose a way to classify the CC Content and Presentation of Evidence elements into five different categories according to the kind of input they require - *Tips*, *Sequence Elements*, *Type-Connected*, *Collective elements* and *Others*. In this thesis, unless explicitly stated otherwise, we call the CC Content and Presentation of Evidence elements CC elements or just elements.

We classify as *Tips* elements that give guidance to the developer but does not require any explicit input in the development documentation, e.g. "***ADV_ARC.1.1.C***-*The security architecture description should be at a level of detail commensurate with the description of the SFR-enforcing abstractions defined in the TOE design document.*"

Sequence Elements are those elements that require an input for different objects of the same type. An important characteristic of this element class is that the corresponding text of the developer documentation can be explicitly partitioned into fragments, each referring to a different object of the type.

We call *Type-containing* elements that are supposed to hold all existing objects of a specific type. We would usually determine one such element per type, most probably the element that requires a definition or introduces all objects of the type.

Collective elements have their content further detailed in other elements. They are in a way similar to the *Tips*, but should be given as an attribute a list of those other elements that further detail their content. Dependence on a collective element means also a dependence on all its attributed elements.

Finally, *Others* refers to any other kind of elements that is none of the above.

The classification of elements determines additional consistency constraints on the developer documentation. For every CC element proper input according to its type should exist in the documentation, for completeness and therefore consistency to be ensured. Dependencies together with CC elements classification and decomposition into objects, provide a strong background for capturing and preserving document semantics. In order to be able to propagate changes, we explore dependency links further.

Chapter 5

Relations

5.1 Dependencies and Hierarchical Relations between Components

Component to Component relations are the only ones made explicit by the Common Criteria document. As can be seen in Table 5.1 where two security functional components are presented, for every functional or assurance component in both CC Part 2 and Part 3, a list of the related dependent or hierarchical components is provided.

FIA_UAU.2	**User Authentication before any action.**	
	Hierarchical to:	FIA_UAU.1 Timing of authentication
	Dependencies:	FIA_UID.1 Timing of identification
FIA_UAU.2.1	The TSF shall require each user to be successfully authenticated before allowing any other TSF-mediated actions on behalf of that user.	
FIA_UID.2	**User Identification before any action.**	
	Hierarchical to:	FIA_UID.1 Timing of identification
	Dependencies:	No Dependencies.
FIA_UID.2.1	The TSF shall require each user to be successfully identified before allowing any other TSF-mediated actions on behalf of that user.	

Table 5.1: Security Functional Requirements

The component dependency information is most often utilized in two situations. First, in the creation of a Security Target, the user has to include security functional and assurance requirements to match their needs. If a CC component A is included in a ST or a PP and according to the CC, it depends on a CC component B, this means that there is a

consistency-related necessity to also include B in the documentation. Second, changes occurring to the contents relevant to B are likely to induce necessity of change in the content of A.

The A Hierarchical To B relationship between two CC components A and B means that the two CC components share a common focus, but the former provides more security and can therefore be used to substitute for the latter whenever necessary. Hierarchicity will also be used to help create Security Targets and Protection Profiles and to ensure their consistency. If a CC component A is hierarchical to a CC component B, this means that A can be used in the Security Target and Protection Profiles at the places where B is required, without loss of consistency. Preserving Component to Component relationships is an important step towards having consistent security documentation.

Further detail can be provided as a continuation of our running example from Section 1.2.

One of the security objectives of the database system is to allow only authorized users to have access to the data of QuickQuery. Therefore, a suitable security functional requirement for the database documentation will be FIA_UAU.2 - User Authentication before any action, which is presented in Table 5.1. As one can see, the CC defines a dependency of FIA_UAU.2 on FIA_UID.1 - Timing of Identification. Therefore the latter should also be included in the documentation. Table 5.1 presents FIA_UAU.2. According to its definition, this component is hierarchical to FIA_UID.1. Therefore, developers could prefer to provide further security and choose to inclde FIA_UID.2 rather than FIA_UID.1. Nevertheless, the dependency is preserved, as the two components are hierarchical to one another.

Table 5.2: Continuation of Running Example

5.1.1 Dependencies between elements

As already specified in Section **??**, we expect to obtain more accurate change propagation by propagating changes between the most granular substructures possible. Therefore, we do not use component dependencies for change propagation, but claim that whenever component dependencies exist, they can be decomposed into dependencies between elements.

CC elements are the structural units that serve as the basis for relating the Common Criteria and the security documentation. It was explained in Section 4.4 that input is provided per CC Element depending on its type classification and in Section 6.3 that the structure of each CC Element serves as imposing consistency conditions in the documentation. Identifying all dependencies between elements represents manual work, that will be done by refining the already defined in CC dependencies between components. Figure 5.1 shows an example how a component to component dependency is decomposed to a dependency between elements.

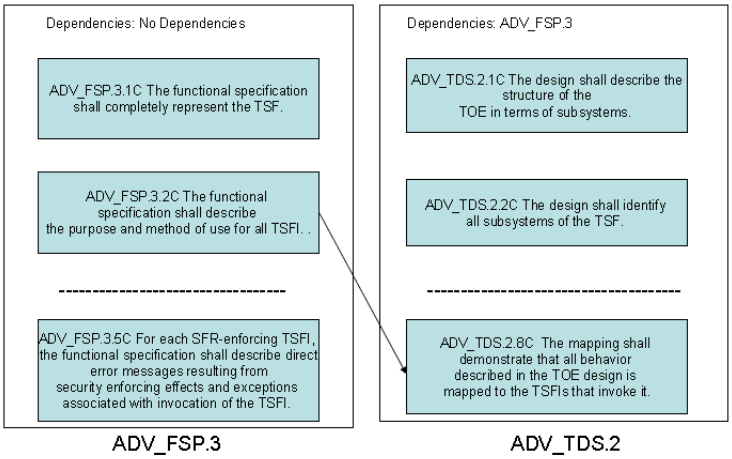


Figure 5.1: Component Dependency Decomposition

The component ADV_TDS.2 and ADV_FSP.3 consist of eight and five elements respectively. Figure 5.1 illustrates how the dependency between them translates to a dependency between just two elements. For compactness, some of the elements are omitted from the figure. This is represented by the dashed lines.

Note that by decomposing dependencies in such a way, a lot of ambiguity in change propagation will be eliminated. As the average number of CC elements per CC components can roughly be estimated to be five, CC component to component relations may relate quite big amounts of information. When decomposing, such a relation, it reduces to just two or three element to element dependencies, which will considerably improve the precise estimation of a scope for change propagation.

The dependency between CC structural can be simple dependencies or can be further decomposed into relations between the element substructures, i.e objects and groups of objects as illustrated in Section 5.2. A simple CC element dependency is shown in Figure 5.2. It gives the information that a CC element A depends on a CC element B which means that the textual content corresponding to A depends on the textual content corresponding to B, i.e. if B's text is modified, A's is also subject to modification.

Dependencies are characterized by their direction. There can exist unidirectional and bidirectional dependencies Changes will be propagated only if the direction of the dependency link matches. If A depends on B, A is influenced when B is changed but not vice versa. Transitivity of the dependency links is another property. Transitivity would mean that, if A depends on B and B depends on C, then A depends on C. If transitivity holds, a change in C will be propagated to both B and A. If it does not hold, however, if B is not modified as a result of the change to C, then the change is not propagated any further to A. Transitivity of the dependency links is left outside the scope of this thesis and needs to be explored in future.

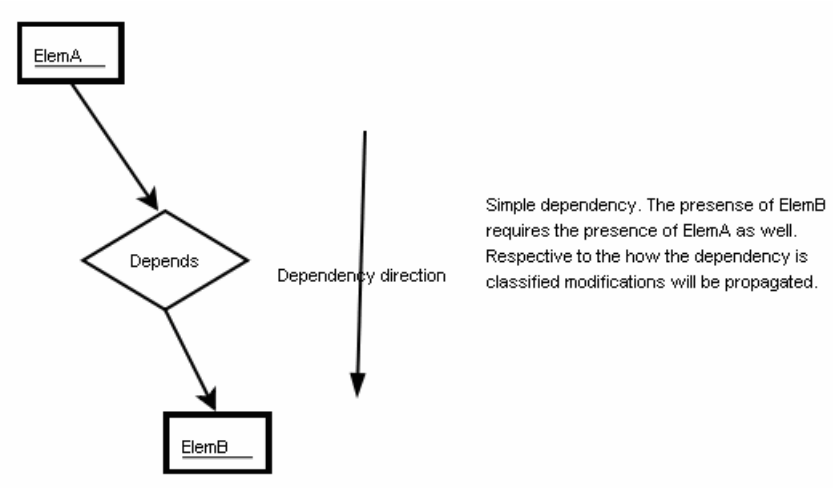


Figure 5.2: Simple Element to Element Dependence

5.2 Structural Relations

Structural relations represent a correspondence between objects of a type. Objects are the structural units of the Common Criteria, as we introduced them in Section 4.2. On a conceptual level, structural relations are similar to binary relations as defined in Set Theory. A relation R between the set A and the set B is the set R of pairs (A,B).

Structural relations in the developer documentation are declared between two sequences or between a sequence and a property. A sequence does not contain in itself real CC objects but references those that exist in the evaluation evidence. We call a maximum sequence one that references all existing objects of a type,

The structural relations can be characterized by their object multiplicity. The multiplicity impose requirements on the participation of objects in the structural relation. All objects referenced by the sequences that declare the relation are entitled to participate in it. The multiplicity denotes the minimum and maximum number of required participations.

We identify one-to-one relations, meaning that for every object X of sequence A, there exists exactly one object of sequence B that relates to X; one-to-many relations that mean that every object of sequence A has to participate in a relation with at least one object of sequence B; all-to-all relations that require all objects from both sequences to participate in the relation at least once; and many-to-many relations that do not define any specific requirements on the number of participations in the relation.

5.2.1 Relations between Sequences

A sequence-to-sequence relation is a binary relation, specified by the object types of the sequences involved, its multiplicity and direction and the name of the CC element that defines it.

Figure 5.3 shows an m:n relation between two sequences. The arrows in the figure

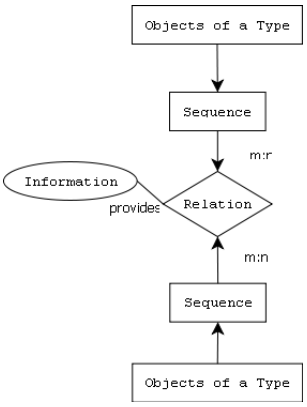


Figure 5.3: Sequence to Sequence Relation

represent dependencies. The two sequences consist of references to objects of a certain type, therefore they depend on the objects. The relation itself is represented as a set of relation pairs and some additional information relevant to each pair. As the relation pairs reference the objects once again, they depend on the object sequences and also on the objects themselves. The m:n multiplicity of the relation tells that from every sequence, at least m and at most n objects should participate in the relation.

To illustrate sequence-to-sequence relations better, we extend our running example, as shown in Table 5.3.

5.2.2 Relations between Sequences and Properties

We identify the notion of object properties. Properties of objects are also represented as objects. Properties differ from the other types of objects in that we are not interested in the group of all objects for a certain property. They do not exist or would not be uniquely identified on their own, but only together with the other objects they clarify. Properties are also defined in the CC elements and are relevant to all objects of the same type, or to a sequence of objects. Another difference between properties and sequence objects is that the former lack a main element, where all objects of the type are listed.

An example of an element defining a sequence with properties is **ADV_FSP.2.2C** listed below.

ADV_FSP.2.2C The functional specification shall describe the purpose and method of use for all TSFI.

This element requires the sequence of all Target Security Functionality Interfaces (TSFI) to be identified along with the purpose and methods of their use. Here the purpose and methods represent two properties. Therefore the relation can be split in two structural relations - one between the TSFI and the methods and one between the TSFI and the purposes. Another option is to considere purpose and methods as one single property.

The developers of QuickQuery security documentation have chosen the "ADV_FSP.3 - Functional Specification with Complete Summary" component as part of their assurance package. Therefore, for consistency reasons, they are prompted to provide input in the security documentation relevant to the requirements of the CC elements of content and presentation of evidence.
TOE Security Functionalities (TSF) consists of all parts of the target of evaluation that have to be relied upon for enforcement of the Security Functional Requirements as introduced in Chapter 2. [Com02d].
The Target Security Functionality Interfaces (TSFI) consist of all means for users to invoke a service from the TOE Security Functionality (by supplying data that is processed by the TSF) and the corresponding responses to those service invocations. The TSFI for QuickQuery is presented by the login prompt that the database provides for user authorization.
The CC Element ADV_ARC.3.7C provided below can be classified as a sequence-to-sequence element.
ADV_FSP.3.7C- The tracing shall demonstrate that the SFRs trace to TSFIs in the functional specification
It defines a sequence-to-sequence relationship between the sets of Security Functional Requirements and of Target Security Functionality Interfaces, both previously defined in the QuickQuery documentation.
This relationship has an all-to-many multiplicity, as all TSFIs shall be addressed by a SFR and all SFRs should relate to a TSFIs.
The developers of the documentation have to provide the relevant information as dependency pairs of objects of the relevant types.
A TSFI present for QuickQuery is the login prompt. On the side of the Security Functional Requirements, there are FIA_UAU.2 and FIA_UID.2, which are present in Table 5.1. Both SFRs relate to the login prompt.
ADV_FSP.3.7C_RelPair1: Login prompt and FIA_UID.2. The functionality of identifying the user is invoked through the login prompt.

ADV_FSP.3.7C_RelPair2: Login prompt and FIA_UAU.2. The functionality of authorizing the user is invoked through the login prompt.
Therefore, the set of relation pairs consisting of RelPair1 and RelPair2 is added to the evaluation evidence in correspondence with the structural relation identified in ADV_FSP.3.7C.

Table 5.3: Continuation of Running Example

The participation constraints of the relation require that the sequence of all objects of type TSFI participate in the relation at least once and that at least one purpose and method is defined for each TSFI.

The relations between sequences and properties define new dependencies between ob-

jects. Clearly the property of an object is dependent on the object and is subject to modification in accordance with it.

5.2.3 Higher Order Relations

Sometimes the Common Criteria elements do not define a relation between just two object types. When more than two object types are present in a relation, we talk about higher-order relations. In our system, these are obtained as the composition of binary relations. The definition of the structural relation that participates is viewed as a sequence of relation pairs and relations are defined as a set of pairs of a relation pair and an object. A higher order relation is, therefore, a binary relation defined between a sequence of relation pairs and a sequence of objects or object properties.

We can again distinguish between sequence to sequence and property to sequence relations. However, here one of the sequences is composed of relation pairs. Using the generalization of defining relations over relations makes the definition of ternary and higher order relations possible.

A type of composite relation requires that a relation is defined over a sequence of objects filtered by a property(the value of another object), i.e. an object of a CC type participates in the relation only if a property holds true for the object. If the object has a specific property object, the higher order relation assigns another object to the relation pair, thus forming a ternary relation. The table below provides an example to illustrate the idea.

The following element is part of the included by the QuickQuery developers security requirements.
ADV_FSP.2.5C For SFR-enforcing TSFIs, the functional specification shall describe direct error messages resulting from processing associated with the SFR-enforcing actions.

This content and presentation of evidence CC element defines a relation between the Target Security Functionality Interfaces which degree of enforcement of security functionalities is SFR-enforcing, i.e. they have a property valued SFR-enforcing, and the SFR-enforcing actions. This dependency also gives additional information about direct error messages.

A composite relation, that requires a property to be defined for every relation pair is shown below. In this relation, properties are defined for the composition between an object and its property, depending on the value of the property.

ADV_TDS.2.6C The design shall summarize the behavior of the SFR-supporting subsystems.
In the dependency defined by ADV_TDS.2.6C, the systems are filtered on the property of supporting the Security Functional Requirements. For subsystems with such property, a summary of the behavior should be provided.

Chapter 6

System Methodology

In the previous chapters, the structural semantics of the Common Criteria and the evaluation evidence were introduced. In this chapter we present our approach to turn the developer documents into fully-fledged document graphs by incorporating the structures within them. Once the security documentation is aware of its own semantics, the problem of management of change translates to the problem of restoring consistent state of the documentation, after a change has been applied. In order to be able to analyze the impacts of changes we describe the consistent state of the documentation and formulate the consistency constraints.

This chapter also contains information about the methodology used to derive the features and functionalities of the proposed change management system. Based on the information we have so far, we reverse engineer requirements for and properties of a software tool that supports the creation of evaluation evidence enriched with structural semantics.

6.1 Semantic Annotation

In Section **??**, the CC structure was clearly identified, but in order for this structure to be useful, it should be made explicit and available for reuse. As a first step, all Common Criteria structures, relationships and their properties should be provided in an explicit form. This is achieved by annotating the documentation with structural semantics.

[DM07] defines semantic annotation as the task of assigning to the entities in the text links to their semantic description. Semantic annotation is a smooth traversal between unstructured text and available relevant knowledge. The semantic data on top of the documentation is what makes the process of semantically aware change management possible. It helps derive information about the range of occurrence of a change and the transitively affected by the change document fragments. In general, semantic annotation may also play an important role in enabling many additional applications, such as highlighting, indexing and retrieval and categorization.

Structural semantics, as defined for our system, are represented by tags in the XML format. The patterns for the structure and content of the documents in the XML format

are specified in the compact syntax of the RELAX NG schema. The RELAX NG Schema for the Common Criteria and the security documentation are derived directly from the structures of the documents as defined in Section **??** and Section **??** respectively. The RELAX NG schema for the two type of documents are not included in this document to avoid repetition of information. What is peculiar about the XML form of the developer documentation, is that it contains an XML element, called "declarations" that contains the declarations of all CC structures included in the security documentation.

Semantic annotation of the IT security documentation can be performed manually. This, however, is a time-consuming and laborious task, which certainly cannot scale to the demands of most real-world applications. Moreover, the annotations will be meaningless without a system that can interpret them in a way that change detection, impact analysis and propagation are enabled. Therefore, it is desirable that a semi-automatic, if not a completely automatic solution is provided.

The solution is provided in the form of a software system that interprets the CC structures and translates them to requirements for and structures of the security documentation. Based on the knowledge about the CC structures, the system must serve as the means to create and simultaneously semantically enrich the content of the documentation. The system underlying algorithms should make use of the provided semantics and ensure the consistency of the created with it documentation. The system's functionalities are determined by the characteristics of the different types of documentation that are to be created with its help.

6.2 From Structures to System Datatypes

The system we propose supports the creation of semantically enriched security documentation and at the same time provides functionalities for the maintenance of documentation. The application parses the annotated Common Criteria document and creates an internal representation of the information. For every CC structural entity type, such as CC class, CC family, CC component etc. a corresponding datatype with the same name exists in the internal memory of the system.

When a CC structural entity is selected as relevant to the documentation, it is mapped to a corresponding security documentation entity declaration.

A large amount of information needs to be stored in the memory of our proposed tool. Many questions appear regarding the storage and access of CC data in program memory and also the invisible representation of the data within the structure of the document.

subsectionConventions for Unique Naming Unique naming is an issue which is compounded when large amount of information needs to be stored and identified. In order to maintain fast access to the structural units and to determine an object in the system unequivocally, unique names should be provided for every structural unit within the group of all units of its type, i.e. there should not be two SAR classes, or two SAR components or so on that are named the same. This ensures that all structural units in program memory can be accessed within their type and name as the key attribute pair.

The predefined in the Common Criteria structural elements already have unique names. The developer, however, may need to extend the already existing structures with further semantic information. In such situation, the options should be possible. The first option should allow the user to specify the name of the entity they create. In case this option is used, extra care should be taken that the user-defined name is unique.

The second option is to provide the structure names automatically. In this case, consistency is ensured by an internal naming-convention algorithms. Overall, unique nomenclature can be achieved by naming the structural units using their location on the data tree. For every edge traversed, to reach a structural unit, a corresponding suffix to the name of the CC unit at hand is added. Therefore, the name of the element, for example, will be a combination from the suffix for the CC class plus the suffix for the CC family, CC component and the CC element number. Similar naming is already used in the Common Criteria document itself.

Another possibility will be to combine the data type and the name of the entity and ensure that every entity is uniquely denoted amongst the entities of its type. This option is chosen for our system due to its simplicity.

6.2.1 Data in Documentation

To relate structural entities to the written text, we need to provide invisible semantic annotations. The opening and closing tags of the XML elements should be stored in information carriers that stay persistently with the written text and are invisible to the readers. This information carriers should also change their position in the text, when new information is inserted or deleted, this is, they need to shrink or expand together with the text. Therefore, we chose to use bookmarks as the structures to create the correspondence between objects in program memory and textual fragments. An opening and ending bookmark related to a structure will denote that the text in the range of the bookmarks will constitute the input for the respective structural entity.
Therefore the names of the bookmarks should determine the structural element unequivocally. As the system provides unique naming convention per CC Structure, the type of the structure as well as the name of the structural element may be used to form a unique name of the bookmark. Because, there can be more than one bookmark per structural element in the text, we use the number of the bookmark as a further identification.
A general name for a bookmark will look as shown below:

”s_n_elementName_t_elementType_c_numberOfBookmark”.

Here ”s” stands for start, i.e. an opening tag, ”n” stands for the name, ”t” for the type and ”c” for the the count or the number of the bookmark out of all bookmarks referring to the particular CC structural unit. Therefore a typical example for a starting bookmark name will be

"s_n_ADV___FSP_1_3_t_ac_c_2".

This bookmark will denote the second bookmark for the security assurance component ADV_FSP.2. Algorithms to translate between program data names and bookmark names are provided within the system.

6.3 Consistent State of the Developers documenation

When we talk about consistent state, we restrict our analysis to a system comprised solely of the CC-based IT security documentation enriched with structural semantics. We can ignore the CC document, once all its relevant semantic information has been extracted and translated to a semantic model for the security documentation. In general, we define the following consistency constraints for the security documentation.

In a consistent state of the Developer Documentation there must be no dangling dependencies for functional and assurance components in the Security Target. For example, if a component A depends on a component B, if A is part of the Security Target, either B or a third component C that is hierarchically higher than B is also part of the Security Target.

Moreover, for every element from the Common Criteria, that is not classified as a Tip or a Collective element and that is part of a specified in the Security Target component, there corresponds a textual input in the assurance documentation. Aside from the text, the textual fragments should consistently include input for the subunits contained in the element at hand. If the element defines a collection of objects of a type (i.e. is supposed to contain the maximum set of objects of a certain type) then the documentation indeed should relate such a collection to the element. The same holds true to relations. For every structural relation defined by an included element, a consistent set of object pairs should be defined to instantiate it.

Another important aspect of system consistency includes consistency of the substructural relations defined in the system. For a structural relation to be in a consistent state, the required relations among the objects of its defining types should be preserved. Meaning the relation needs to be instantiated by a set of object pairs and the multiplicity of the participating objects should be consistent with the multiplicity property of the relation. Also all objects taking part in the relation need to be present and of proper type in the documentation. For example, if a one-to-one relation F exists between two object types A and B, a consistent state will mean, that for each object x of type A there exists a unique object y of type B and the pair (x,y) is part of F.

Existence of no contradictions is another consistency constraint. The contradiction we intend to handle deals with an incoherent naming of the same artifacts throughout the documentation. Every artifact mentioned in the security documents should have a unique name that does not change through the different document parts. We claim that in a consistent state of the security documentation the sufficiency of countermeasures is ensured and the proof of correctness is easier to achieve. Consistency is ensured because in order to have a consistent state, we need all structures in the system to be in a consistent relation to

one another. This means, for example, that for every identified threat and security objective there exists a proper countermeasure. Translated to the language of the Common Criteria it defines consistency of countermeasures. Once sufficiency and consistency of the system are ensured, the evaluation of the correctness becomes an evaluation of the meaning of the documentation.

6.3.1 Restoring Broken Dependencies

Existence of a broken dependency will mean that at least one of the consistent state conditions as introduced in the previous section is not fulfilled. Depending on the type of the broken dependency link, we propose possible steps to fix it. Right now we have determined that inconsistent state may appear as a result of one of the following actions: deletion or addition of a new component, deletion or addition of a new object of a certain type, modification on the relations between existing objects and possibly some more that are going to be identified further in our work.

If a component gets removed from the system, e.g. from Security Target, all components that directly or indirectly depend on it will have to be removed from the system. Also the corresponding inputs in the TOE documentation for the removed components should be removed. If a component is added, then all components that it depends on, should also be added in order that the system returns into a consistent state. Similarly upon addition of an object of a certain type, the relationships with all other object types for this type should be checked, and when a relationship is broken, actions should be taken for restoring it. For example, adding a new threat will require adding a new countermeasure, this at hand will require relating this countermeasure to a security functional requirement and so on.

On the side of the TOE documentation, all removal of text fragments corresponding to an assurance element inferred from the Security Target will leave the system in an inconsistent state, that will not be fixed until further input. Addition of further information, modification or deletion of text fragments that are not subjects to any dependency, will not change the system consistency state. Addition of dependent information, as well as any other significant modification of fragments of the developer documentation, may or may not leave the system in an inconsistent state. Nevertheless, when a change leaves a system in an undetermined state, the change should be propagated thus providing the option to restore any broken dependency that may occur on the propagation path.

6.4 Building Developer Documentation

There are three types of developer documentation: Protection Profiles, Security Target and TOE documentation. As the TOE documentation differs significantly in its essence and purpose from the rest, we provide two major functional modules for our tool. The two modules are highly interrelated to the point that the Target of Evaluation makes references to and builds dependency on the structures of the Security Target. Therefore we will

consider the two modules together and will make distinction to only those characteristics that actually differ.

Before we take a closer look of the properties of such an application system, we would give an in detail introduction to how the CC structures get translated to security documentation structures.

6.4.1 Building Protection Profiles

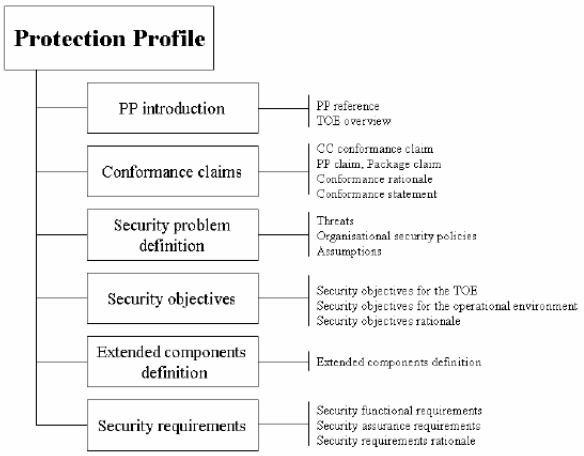


Figure 6.1: Protection Profile [Com02b]

Figure 6.1 represents the backbone structure of a Protection Profile (PP). This document type was introduced in Section 2.4.

In this thesis, we do not aim to provide support for the creation of Protection Profiles, but would rather support their integration with the rest of the documentation. The reason behind it is twofold. On one hand, Protection Profiles are usually not created by the user. Already existing Protection Profiles from special catalogues are reused. The ability to reuse Protection Profiles is, therefore, more valuable than to create them. On the other hand, when the structures of Protection Profiles and Security Target are compared, a lot of similarities can be found. The Protection Profile sections represent a subset of the sections of the Security Target. The object types defined in the two kinds of documents and the relations amongst them are mostly the same. Therefore, we can easily at later stage transform the tool provided in Section 6.4.2 so that it becomes suitable to build both Security Targets and Protection Profiles. As a result, we direct our efforts at creating support for building only Security Targets and assurance documentation and at including the knowledge about the Protection Profiles therein.

6.4.2 Building the Security Target

The structure in Figure 6.2 represents the tree view backbone of the ST documentation. This is also the structure that appears to the left, in the tree view window of the tool interface and is used for easier navigation through the ST document. Every leaf of the represented tree view has a corresponding text fragment in the Security Target document.

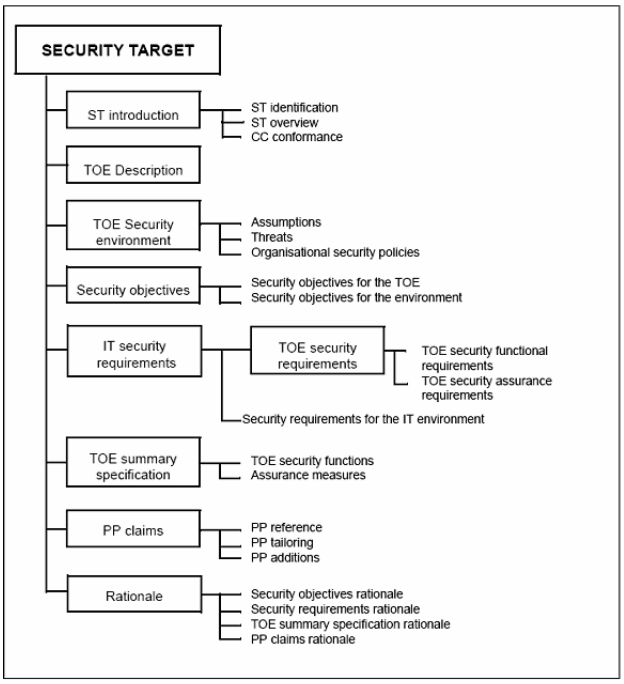


Figure 6.2: Security Target Content

Therefore, the Security Target becomes the reference document for all the other provided documentation and hence for the whole process of evaluation. This means that the Security Target document should exist and be consistent before any progress can be made in creating assurance documentation.

Once the Security Target document is created, its structure needs to be available as a reference to develop the Target of Evaluation. Therefore the CC structural units relevant to the Security Target are the same as those for the rest of the security documentation.

Due to the generally static structure of the ST documentation and its limited, predefined number of different object types and relationships between them, the ST is easier to create than the other documentation tools. Moreover, for the creation and change management of Security Target documentation, enough information is provided in Part 1 - Introduction to the Common Criteria. This means that in order to create a Security Target no semantical annotation and automatic loading of the CC Part 3 will be necessary.

The ST-related functionalities of our systems are a subset of those required to create

assurance documentation. From implementational point of view, it is easier to first create the functionalities to build Security Targets and then extend them to build the rest of the documentation.

To provide more insight about the necessary steps to create Security Targets, we extend our running example as shown in Table 6.1

The developers of QuickQuery use the CCWord application to define the Security Target for their system. Therefore they need to define all structural units required by the Security Target. First of all, the developers should identify the threats to the database. The developers will insert "T1" and "T2" as shown below and annotate them as objects of type threats. Threats: T1. Access to functions or data of QuickQuery by persons who are not authorized to use QuickQuery. T2. Access to functions or data of QuickQuery of an authorized person in an unauthorized way. Every threat that gets inserted in the Security Target, puts the document in an inconsistent state, because it creates unsatisfied relations. As derived from the Common Criteria the following structural relation should hold for the Security Documentation: R1: Every threat should be countered by at least one security objective. Therefore, to restore consistency the users are required to define objects of the security objectives type and later link these objects to the threats, so that all threats are countered. The user annotates the following security objectives for the TOE: 01.The TOE must limit the access of people to only those who are authorized. 02.The TOE must provide only the relevant data for every registered user. CCWord would then enable the users to link every threat to its corresponding security objectives and thus create the so called set of Relation Pairs. Therefore R1 will be instantiated and satisfied by linking T1 to O1 and T2 to O2. After the linking has been done, the system is again in a consistent state. The user will continue writing the ST document until the moment it is complete and consistent. The Security Target document will be complete once data is provided for all required by it CC Structural Units and consistent once the dependencies on these structural units are satisfied.

Table 6.1: Creation of Security Target-Running Example

6.4.3 Building TOE Assurance Documentation

For this section we will again consider Figure 4.1 and Table 4.2. An important characteristic of the tool for TOE development is that it builds on top of the Common Criteria annotated with the proper structural semantics and the Protection Profile and Security Target as defined by the developer with the help of our Security Target tool. The TOE documentation consists of inputs for every single content and presentation of evidence element of the predefined assurance components. The relationships between the inputed text fragments will be the same as the relationships between the respective assurance elements. The complexity of building assurance documentation comes mainly from the amount of information that should be previously extracted from the CC Part 3 and reused, as well as from the generally heterogeneous structure of the assurance documents.

The principles of creating assurance documentation completely coincides with the general principles of creating security documentation that were discussed in this thesis so far. The methodology to create assurance documentation is therefore further detailed in Chapter 7.4.7 where the CCWord system is introduced.

Chapter 7

CCWord

The primary task of this thesis is to solve the problem of management of change in IT Security Documentation based on the Common Criteria. Management of change in IT Security Documentation is a complex problem. To solve it, we design and develop a system that would at the same time support the creation and maintenance of semantically-enriched security documentation. We call the system CCWord - Word for the Common Criteria. The goal of CCWord is to automatize the process of creation and maintenance of IT Security Documentation. It also aims to reduce the time and effort necessary to provide sound, consistent and semantically enriched documentation of the security functionalities of IT systems.

This chapter describes the design decisions made to build CCWord, whose capabilities and functional requirements are listed in Section 7.2. These design decisions lead to the architecture presented in Section 7.4.

7.1 Design Goals

By creating CCWord, we pay attention to design features such as reliability, ease of use, upgradeability to newer versions of the CC and interactivity.

The reliability of the tool is achieved by the reliability of the underlying Microsoft Word platform and by ensuring the deterministic behavior of the system underlying algorithms.

CCWord needs to be easy to use, in the sense that end users that have little experience with the Common Criteria are also able to benefit from the functionalities offered by the system.

Upgradability is a property that directly relates to change management between versions of the CC. This design goal is possible, because our tool will load up automatically on the basis of pre-extracted information about the Common Criteria structural units. Because the newer version of the Common Criteria differ from the previous ones only in their content but not in their structure, the knowledge obtained about the CC structure will remain the same. Thus the problem of upgrading the tool to new versions of CC reduces to the problem of annotating every newly appearing version consistently with the created

RELAX NG Schema and updating accordingly the system configuration files.

The developers are the entities in charge of the documentation and they usually know much more about the existing dependencies between different parts of the documents than they can express in the text they write. That, and the fact that there may exist inner-document dependencies in addition to those that can be derived from the CC serve as our main motivations for the interactivity property. Our aim is to help the developer provide further semantics than what can be inferred from the CC. The tool allows for the creation of new object types and relations in between object type references and in between text fragments.

Further interactivity is achieved by enabling the developer to classify changes made to documentation, request or stop the propagation of a change or to request consistency checks. Also, it is the job of the system user to decide how to integrate a change.

7.2 System Specification

In this section, we present the functional and behavioral properties of CCWord. The main purpose of CCWord is to create semantically-enriched, maintainable IT security documentation with less time and effort. This aim will be achieved through the interaction of a wide range of features and functionalities. For each system feature described, a conceptual-level explanation of how it is achieved is provided.

7.2.1 System Initialization

First of all, the software system needs to be informed and able to reason about the structure of the Common Criteria, as it is defined in Chapter 4. The information about the CC structure is provided from configuration files in the comma separated value(CSV) format that the CCWord system loads upon start up. These configuration files, are currently manually obtained from an XML annotated version of the CC. They represent a compact representation of the CC structures in a format that is easier to parse by our application.

The underlying functionality of the CCWord parses the provided information about the structure of the Common Criteria and transforms it into internal data, guidelines and requirements for creating consistent documentation of the security properties of IT products. The users of CCWord are supposed to select those CC Structural units that are relevant to their documentation. The structural units selected by the user impose conditions on the content and consistency of the security documentation, as described in Section 4.1. Therefore, the users should provide enough information to satisfy the imposed conditions.

7.2.2 Creation of Content

One of the characteristic features of the CCWord system is its interactivity with the user. Although the goal of the system is to reduce the manual work and provide for semi-

automatic document creation, the user interaction with CCWord remains the fundamental means to create content, enrich it with semantics and ensure document consistency. User participation is required through all possible steps. The prompts of the CCWord interface users aim to guide the user to make best use of the relevant information at their disposal.

The CCWord system allows the user to insert and format content in the security documentation. As mentioned in the previous section, once CC structural units are selected as relevant to the security documentation input should be provided and annotated to satisfy the imposed consistency constraints. With the help of CCWord, text fragments can be either inserted relevant to some existing CC Structural Unit, or plain text can be inserted in the documentation and consequently annotated with semantics.

The options of content formatting that CCWord provides are inherited from its host application - Microsoft Word.

7.2.3 Navigation

For CCWord interaction with the user is of primary importance. On one hand, the system guides the user in the process of documenting the security properties of their IT products. On the other hand, the system cannot operate properly without a prudent participation on user's side.

The system guides the user in several different ways. Guidance at the beginning of document creation is represented by providing a document template with a format that, if followed, leads to the development of consistent documentation. The template provides the structure of the document that needs to be created. With its help, the user will know how to structure the information they want to provide. This feature diminishes the amount of preparatory work the user needs to undergo before they can start writing proper security documentation and moves the focus of the document creation process from the structure and representation to the content of documentation itself. A more formalized common language between developers and evaluators becomes available as an additional advantage of using a standardized template.

Currently, the initial template that the CCWord provides is relevant for writing Protection Profile and Security Target documentation. As these two document types have a fixed structure their sections can be automatically created. For more detail on the methodology used to create PPs and STs refer to Section 6.4.2 and Section 6.4.1 respectively. A template for writing assurance documentation would be considered more as a suggestion than as a rule. A template can be provided per the necessary textual content for a security assurance component. Adding this functionality to CCWord will be left as future development.

Another form of guidance available to the user is navigation. Navigation is accomplished either through a treeview model or through a navigation panel. The treeview model gives a compact representation of the documentation. Its nodes represent properly-nested sections, subsections and other available in the documentation structural entities. The treeview is considered part of the CCWord interface and currently exists for the Security Target documentation, as visualized in Figure 7.6. When a node from the tree view is selected, its corresponding text fragment is highlighted.

The navigation panel provides drop down menus containing the structural units present in the documentation as shown in Figure 7.7. For every entity present in the CC-based documentation, the corresponding text fragments that give information relevant to the structure can be highlighted upon request. In this way an easy navigation is ensured.

The interface of the tool provides further guidance. For every selected text fragment, the user can request information about the structures that are instantiated through the text or they can view the list of all applicable dependencies over the text fragment. All system functionalities are available to the advantage of the user through the CCWord interface.

7.2.4 CC Structural Entities Extension

CCWord provides the user with the option to extend the set of structural units inside the security documentation with structural units that are not defined in the Common Criteria.

This feature is useful in the creation of Protection Profiles and Security Targets where sometimes components not present in the CC should be defined. It can serve to augment the Evaluation Assurance Level with user-defined security components.

Also, the set of object types and their objects are dynamic and often needs to be manually expanded. Another option for extending the CC structure is the definition of further dependencies. Creation of dependencies will be an eminent feature during the process of security properties documentation. Sometimes, the user of the system would know that one text fragment depends on another and would like to make that explicit. However, this text fragments may not be classifiable as one of the CC structures defined. CCWord would provide the option to classify a text fragment as a general structural unit and to define a dependency link from this general unit to another existing entity.

7.2.5 Documentation Maintenance

CCWord provides the option of consistency check over the created documentation. Consistency check is a feature enabled as a result of the semantic layer added on top of the documentation. It is the means through which the system restores its consistent state as described in Section 6.3. CCWord validates consistency using its underlying algorithms. They go through all relevant CC Structural units and check if input is provided for every one of them in the security documentation. Once this is accomplished, the system examines if the consistency constraints due to dependencies or structural relations are also CCWord allows the user to request consistency over a selected set of CC Structural Units or over the whole documentation.

Performing changes over the documentation can break its consistency. Not every change will have the same impact. CCWord provides the user with the capability to classify the changes made on the documentation in order to determine those that are expected to affect other parts of the documentation.

Another functionality provided by CCWord is the possibility of change propagation. Change propagation comes as a result of specific changes. If change propagation is re-

quested the change propagates along the dependency links coming out of the affected structural unit. Further change propagation is done in an iterative manner. The system tells the user where change may need to occur. Then the user should determine, if they will make further changes. For every change, the users should again classify it and if applicable, propagate it further.

CCWords maintains documentation by creating for each project a folder with project-specific configuration files. Using this files, the system is able re-initialize the documentation in exactly the same state as it was left over by the previous session. Functionally, this means, that all relevant data should be stored in a format parsable by the application and loaded in program memory upon demand.

7.3 Implementation

The tool was implemented as an extension to MS Word 2003 using the Visual Basics for Applications (VBA) programming language. Among the reasons for this choice are the fact that Word, already by itself, is a fully operational environment for documents creation, that provides a wide range of useful custom functionalities. Therefore, creating an add-in for Word enables us to concentrate on the CC-related features of the system, rather than on capabilities related to text editing and formatting.

Moreover Word 2003 and its IDE had been in use for many years before the beginning of this project and had proven to be stable and reliable.

VBA is also a suitable programming language because it enables the CC Security Documentation to be bundled and transported together with the CCWord functionalities. Every security document, therefore contains with itself a copy of the CCWord application, which makes it self-contained and independent of other editing environments than Microsoft Word.

7.4 Subsystems

CCWord is a complex system that results as the interaction of a wide range of functionalities. This section describes the decomposition of CCWord into subsystems specialized at accomplishing a certain part from the main task. CCWord can be logically decomposed into several subsystems as presented in Figure 7.1.

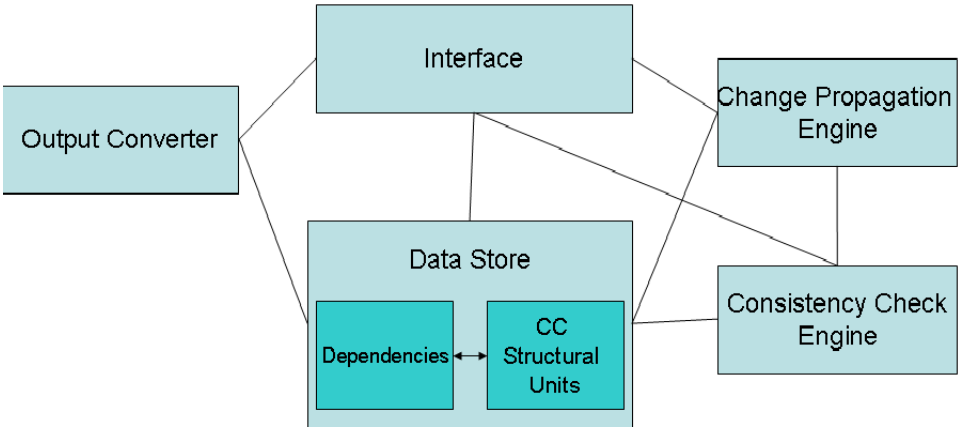


Figure 7.1: CCWord Subsystems

The rest of this section contains a discussion of the CCWord modules.

7.4.1 Interface Module

Defining the GUI features and functionalities is a very important task in the system design with a direct influence on the usability of the system. All relevant functionalities should be invocable in an easy and intuitive manner. CCWord makes its functionalities available through a dockable toolbar in the menu of the host application Microsoft Word 2003. The toolbar appears when a document is opened using the CCWord template. Figure 7.2 shows the CCWord toolbar.

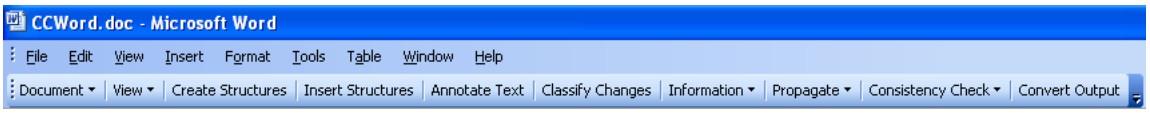


Figure 7.2: CCWord Toolbar

The functionalities provided through the interface fall in one of the following categories: document management, creation or insertion of structures, textual annotations, information request, change classification, consistency check or change propagation. All these functionalities are directly accessible through the toolbar buttons. Most of the names of the command buttons are self-explanatory.

"Document" provides the option to choose the type of document that should be created with CCWord - Protection Profile, Security Target or Assurance Documentation and therefore determine the overall functionality of the system. The "View" popup button is used to select which parts of the interface should be displayed, such as the treeview model

of the document or other possible navigation windows. The buttons - "Classify Changes", "Propagate" and "Consistency Check" are used to request the respective operations.

If the CC Structures have not been automatically loaded within the system, as a result of the selection of the Evaluation Assurance Level for example, they may need to be later included by the user. This is done through a userform for selecting CC Structures, which is shown in Figure 7.3.

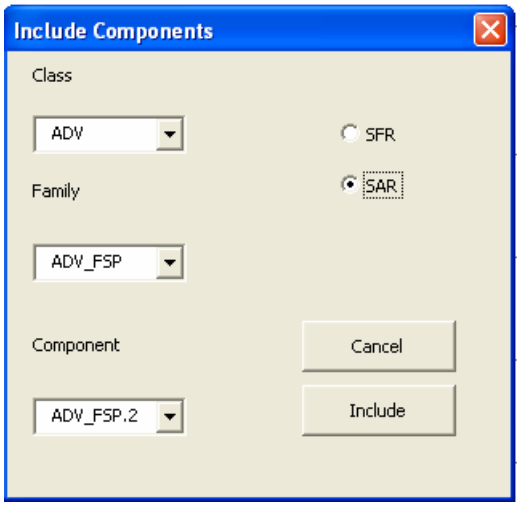


Figure 7.3: Include CC Structures

The smallest selectable structural unit from the CC is the component, therefore the interface provides navigation through all classes and all families to a list of all the components of the selected family. Once a component is selected for inclusion, it is going to be added to the collection of active structural units. Moreover, if it is hierarchical to an already included component, a warning will be send to the user, and todo tasks will appear to help them resolve the conflict. Currently the respective todo tasks highlight the textual correspondence of the component elements and request reassignment of text ranges and deletion of the obsolete structural units.

The user can extend the CC structures present in the system by creating their own instances of particular data structures. This feature is especially beneficial during the creation of the Security Targets, when the CC defined components are not sufficient and need to be augmented. A userform that is used to accomplish that is presented in Figure 7.4. This userform appears on pressing the "Create Structures" button of the CCWord toolbar.

The userform used to provide annotation capabilities is shown in Figure 7.5. Information units for all possible security documentation entities can be assigned text range in the documentation through the use of annotation forms. The user needs to first select the type of object to annotate - component, element, object of a type, a relation pair or a dependency. The relevant parts of the userform are enabled in accordance with this selection. In Figure 7.5, a textual fragment from the security documentation will be linked to

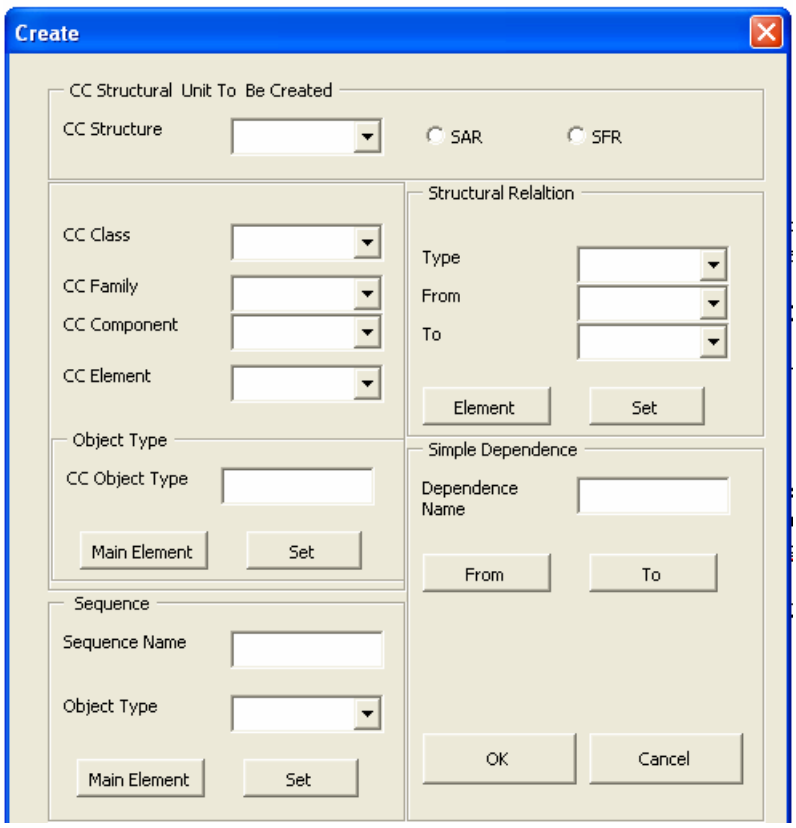


Figure 7.4: Create CC Structures Userform

the CC assurance element "ADV_ARC.1.4C - The security architecture description shall demonstrate that the TSF protects itself from tampering. " With the help of this interface, the user can annotate one entity at a time.

The document management features provided by the interface include the inherited from MS Word capabilities to create, format, save and load documents, as well as the previously described options provided by the "Document" button of the interface.

The interface provides the options of navigation through the documentation. The treeview userform of the ST-related functionality is shown in Figure 7.6. It displays the names of the different sections of the documentation, as well as the relevant security functional and assurance components in these sections. Selecting a component node, highlights its corresponding text fragments in the text.

The treeview may later on be extended to present further nested, more granular CC Structural entities, such as elements, object types and their objects, object sequences, etc.

Another form of navigation represents a recurrent pattern in the interface userforms. It helps the user to find their way through the data structures in program memory. These are dropdown menus that help the user select structural entities gradually. As an example, to reach a specific CC Element in data memory, a userform with four drop down menus is

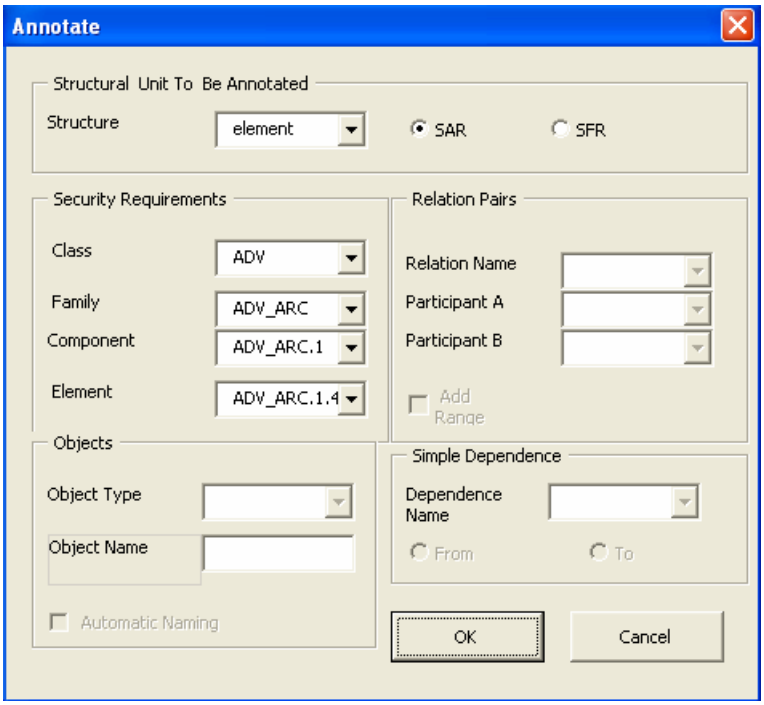


Figure 7.5: Annotation Userform

provided. The user first selects the CC class name, followed by the CC family name, CC component name and CC element name. The number of possible choices for each step is therefore considerably narrowed.

Because the structure of assurance documentation may be quite heterogeneous, we do not provide a treeview representation for this case. Therefore, navigation is accomplished through the userform in Figure 7.7 which uses the above mentioned dropdown menus. The active in program memory data structures will be accessible through the menu items. When an entity is selected, the corresponding text fragments in the security documentation will be highlighted. The other way around is also possible. The user will select a text range from the documentation, press the request information button of the interface and obtain the names and descriptions of the structural units which contain or are contained within the range.

Interface that displays the todo tasks should be provided on the same userform as the main navigation panel for the respective type of documentation. This is due to the idea, that both the navigation and the todo interface should stay visible through the whole process of document creation. Currently the todo task menu is present together with the treeview in Figure 7.6 and is separated from the navigation menu used for the TOE assurance documentation.

The todo tasks interface contains a drop down menu, that displays the names of the tasks that need to be performed to restore the consistency state of the documentation. For

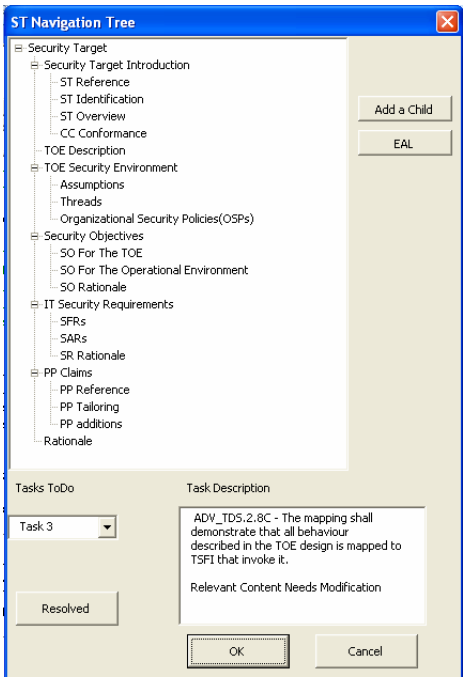


Figure 7.6: TreeView

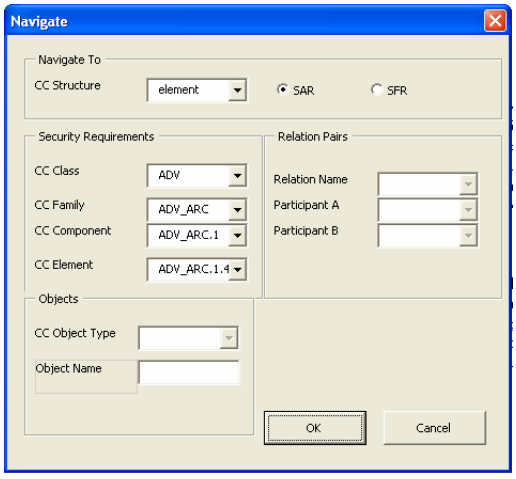


Figure 7.7: Assurance Documentation Navigation

every selected task, its description appears in the text box to the right. The user is then supposed to work on the respective task and once done, mark it as resolved and choose whether to propagate the changes further.

Change classification options are provided to the user as shown in Figure 7.8. When a change to the documentation is performed, this interface module allows the user to

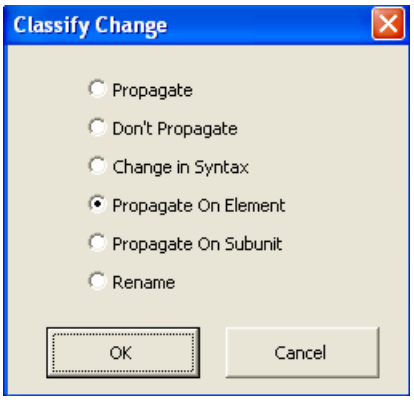


Figure 7.8: Change Classification

classify it as one of the following options: propagation change, no propagation change, propagate on element, propagate on subunit, renaming and syntax change. Depending on this classification the system takes a decision to propagate the change or not. More information is presented in Section 7.4.4, where the change propagation engine is discussed.

Change propagation is currently invoked through the system's interface. When invoked so, it is performed over the dependencies of a selected security documentation structural unit.

Another functionality invoked through the interface is the possibility of consistency check. Three options are provided by the popup menu of the CCWord toolbar, a consistency check for a text selection, a consistency check of the relations defined for a structural unit or a consistency check over the whole documentation.

7.4.2 Structural Units and Dependency Datastore

In Section 4.3, the datatypes defined in the Common Criteria were identified. For this thesis, a self-contained part of the CC document was annotated according to the RELAX NG schema corresponding to the CC data type model present in Table 4.2. The information from this annotated version was extracted and data instances were created and stored in comma separated value formats with predefined structure.

An important part of our task was to create a data store for this information that is easily accessible by the system. One possibility was to connect our application to a database management system and provide for a fast and easy access to the required data. An available database choice was MS Access, because of its inherent interoperability with MS Word. Another plus to this option was the possibility to load only the relevant at the moment of execution data in program memory. This option, however, would have contradicted with our idea to keep the CCWord system and the security documentation coupled and to transport the document and its associated data and algorithms as one single file. Therefore the best possible option was to store all information we need in

meta-variables in the CCWord documentation.

Within the system, we make distinction between configuration data and active data. The former comes purely from the Common Criteria and is loaded when a new document needs to be created or the active data needs to be extended. The latter consists of the part of the configuration data that is used within the security document and of all the other data that is created by the user and is relevant to reloading the documentation in the same state it was left over.

System underlying algorithms are used to save the created data in hidden configuration variables or to access the stored data and load it in program memory. Starting a new project or proceeding with an existing one is then translated into loading a new, initial configuration or restarting the work on an already existing, modified configuration. All other system functionalities operate over the program memory knowledge base, as it contains the only possible and relevant information that can be used to check consistencies, propagate changes and provide for management of change possibilities.

7.4.3 Consistency Check Engine

The consistency check engine is the connection between the program memory knowledge base and the change propagation engine. Its task is to determine whether a consistent state of the documentation as defined in Section 6.3 is ensured.

A consistency check can be invoked over a single structural unit, over a text range or over all active structural units stored in the program memory knowledge base. In the former case, the system checks whether all necessary relations over the corresponding CC structural units are properly defined and satisfied. A dependency is satisfied, if the structural unit dependent upon is defined in the documentation. A structural relation is defined, if in the documentation exists at least one object pair pertaining to it. The relation is satisfied, if the requirements derived from its type and participation constraints are met.

If a consistency check is invoked over a text range from the documentation, the systems detremines the most granular structural entity within the range and then calls consistency check over the entity.

Consistency check over the whole documentation is equivalent to a check over all active data. The algorithm traverses the whole knowledge base and checks whether all required data is defined and all relations are satisfied. Currently, this algorithm performs a brute-force consistency check It exhaustively traverses the knowledge base, by checking if the consistency constraints are satisfied for every included in the documentation CC structural unit. An optimization to this algorithm should be provided as future work.

In cases consistency is not ensured, tasks are created in the todo list of the system.

7.4.4 Change Classification and Change Propagation Engine

The functionalities - change classification and change propagation are closely interrelated. Change classification is interactive and performed by the user. In classifying changes, the

system strongly relies on the user's wise performance and understanding of the documentation. When the user applies a change, they should select the smallest meaningful text range the change applies to, classify the type of change and request change propagation.

There are five possible ways to classify a change. The "no propagation" classification applies for cases when the user knows that their actions will not affect the documentation further. Currently a "syntax change" leads to no propagation. As a future work, a syntax correction of a word can be improved to cause automatic correction of the word through the whole documentation. A "rename" relates to cases when the name of a referenced information unit has been changed. For consistency reasons the name change should be applied everywhere. This feature is also going to be provided to the CCWord in future.

There are three possible types of propagated changes - simple propagate, propagate on element and propagate on subunit. For every change that needs propagation, the most granular security documentation structural unit containing the changed area should be determined. The change is propagated to all dependent entities. If the change is defined as propagate on element, then the respective CC element dependencies will be considered for propagation.

The change propagation engine is invoked whenever inconsistencies appear, a change applied requires propagation or by user request.

Propagation Algorithm Annotation of semantic knowledge in the security documentation converts documents into full fledged document graphs, where the identified structural units represent nodes and the dependencies correspond to edges between nodes.

Therefore, the collection of TOE Security Documentation constitutes a graph containing the graphs given by the annotated Protection Profile, Security Target and Assurance Documents plus a number of additional links corresponding to the interdependencies between those documents and connecting the document graphs. The so defined graph is directed, due to the direction property of the identified Common Criteria relations. The problem with this data structure is that it may contain cyclic paths which can result into infinite loops during change propagation and thus render our system unstable to respect to small changes.

As we wanted changes in structural units to affect their dependent units only once, we had to prevent possible cyclicities. That was accomplished by converting the graph structure to a spanning tree. For a connected, undirected graph, the spanning tree data structure represents a subgraph which is a tree that connects all the vertices together. In our case, however, we deal with directed graphs, therefore we use an approximation of the spanning tree data structure, and use the so called Chu-Liu/Edmond's algorithm [GGST86] to convert between structures. This conversion can be performed without loss of crucial information, as in the process of change propagation dependency links are used only as information carriers, i.e. the message "A change has occurred" needs to be propagated to all nodes in the graph. In terms of node reachability, a spanning tree performs as good as the graph itself, but it performs much better in terms of efficiency. Below we describe the algorithm, that has been used for the conversion.

Chu-Liu/Edmond's algorithm algorithm is an optimum branching algorithm proposed independently first by the Chinese mathematicians Chu and Liu (1965) and then by the American mathematician, Jack Edmonds (1967). The algorithm can be used to determine minimum and maximum spanning trees for weighted directed graphs. We currently consider all edges of CCWord to have the same weight. Therefore, we use an adaptation of the algorithm for our purposes. Change propagation is invoked starting from a single structural unit, or so called node in the document graph. To find a spanning tree, we start with a tree consisting solely of the initial node. Edges are added connecting nodes to the tree. If an edge creates cyclicity, it is erased. This algorithm ensures that all nodes from the graph data structure are reachable from the initial node, because if they are not, an edge can be added to connect them to the tree without creating cyclic path.

7.4.5 Output converter

A drawback to CCWord is its platform dependedance. After documentation has been created using CCWord, a good approach will be to store the data in a standard format that can be read, understood or used by other applications. A suitable format is XML, due to its simplicity and usability. Moreover, the semantically annotated documentation created with CCWord has its direct correspondence in the XML format. The translation to XML format will be compliant with the RelaxNG schema provided for the security documentation. Every pair of opening and closing bookmarks will represent an XML elements. As the bookmark names were unique in the system, they can be used as a distinguishing name property for the XML elements. This module is left as future work.

7.4.6 Protection Profile Annotation Module

Security Targets are usually based on and reference one or several Protection Profiles. The developer should be able to include Protection Profiles in the documentation. Most Protection Profiles are already available on the market and ready for use by the developers of security documentation. A system to include existing Protection Profiles is going to be provided as part of the CCWord application.

Comparing the structures of the Protection Profiles and Security Targets as given in Figure 6.1 and Figure 6.2, we noticed that they share a direct correspondence. Most of the sections of the PP and the ST are the same, both by name and content requirements. Therefore our tool should ensure that the Protection Profile is subsumed in the Security Target. This will be achieved by annotating the structures of the Protection Profiles as structures of the Security Target and creating all required relations amongst them. From a system's view point, the Protection Profiles become a part of the Security Target and benefit from all rules and functionalities that apply to the Security Target. Protection Profiles can currently be loaded in the system annotated with their suitable semantics and included in the security documentation.

7.4.7 Evaluation Support

The main goal of the CCWord system is to support the evaluation lifecycle. Although, its primary focus is on the creation of developer's documentation little can be done without relating the security documentation to the evaluation process and evaluation reports. Therefore we provide evaluation support as part of the CCWord system.

We propose two possible modules for evaluation support. In the first, the evaluators have the possibility to annotate all the fragments of security documentation, where the evaluation determined that insufficient, inconsistent or incorrect information was provided, i.e. all points of evaluation failure. After that the developer can apply consistency check over the concrete points of failure and use the guidance CCWord provides to restore the consistent state of the system. On the developer side, the module also provides the possibility to annotate all text parts that were changed as a result of the necessary adjustments.

Another option can be provided by a CCWord module that serves the purposes of version comparison. This module works as follows. It first compares the set of all available CC Structures to determine all changes in the structures from one version to another. For all the structural units that are the same for both versions of the documentation, CCWord runs "diff-comparison" to determine possible differences in their textual content. After all comparisons have been completed, CCWord notifies the user of their results. Namely, lists of all CC Structures that have been added or deleted from the documentation are provided, as well as the names of those structures the textual content of which differs. For this module to provide reliable results the names of existing in the documentation CC Structures should be preserved between versions.

Chapter 8

Conclusion

This thesis presented a solution to the problem of change management in IT security documentation based on the Common Criteria for IT Security Evaluation. The solution was accomplished in the form of a prototype system, called CCWord that is implemented in Visual Basic for Applications as an add-in to Word 2003. The system assists the creation of semantically enriched, IT security documents that share a common document format. The structural semantics for the documents was derived by a careful analysis of the Common Criteria, the security documents based on it and their corresponding evaluation reports. In addition to the data structures already described in the CC, such as classes, families, components, elements and dependencies, we identified more granular substructures which we called object types and objects. These new structures define or participate in what we called structural relations among one another. Auxiliary data structures such as sequences and properties were also introduced. They help us keep track of consistency constraints derived from the structural relations.

The problem of management of change of IT Security documentation based on the CC translates to the problem of restoring a consistent state after a change is applied. Therefore, we defined the consistent state and the consistency constraints for our documentation.

As a further step, the structure of the Common Criteria and a mapping between it and the security documentation were identified. A software system that we called CCWord was proposed in accordance with the requirements posed by the security documentation properties. CCWord operates over a database of all identified CC structures. For all CC structures selected for inclusion in the documentation, CCWord initializes security documentation entities and consistency requirements in its program data store. The tool provides the option to link respective structural instances to text fragments in the documentation as well as to instantiate dependencies and structural relations. Upon the user's request, the system can execute algorithms to ensure that the consistency of the written documentation is preserved. The algorithms traverse each active CC structure and check if all necessary information units are instantiated and if all the relations defined between them are satisfied.

Change detection, classification and propagation are the operations the system performs to restore its consistent state. Change detection and classification are interactive tasks,

performed by the user. Change propagation is based on the graph created by the structural entities and their dependencies. To avoid cyclic paths, the graph is first translated to a tree data structure. After the areas affected by the change are estimated, it is the task of the user to make all necessary adjustments to the content of the documentation and to thus integrate the change.

Further proposed functionality would be provided by a conversion of the created documentation to an XML format and by a support to the process of evaluation.

8.1 System Limitations and Future Work

Limitations of the CCWord system can be generally classified as two types - due to the limitations of the host application and the implementation language and due to the system design.

CCWord currently exhibits several limitations due to its host application. A drawback in the interface design appears because VBA does not allow for the creation of customized task panes. Therefore, the treeview is not docked to the side of the interface as desired, but is used floating over the main interface window. Another limitation, is the impossibility to implement event triggered change detection for the documentation. Currently no change detection is performed, but the user manually sends the "change performed" message. Both of these can be improved in future versions of the system that use more advanced host applications. Transporting the functionality to Word 2007 and its Ribbon system already provides solutions to these problems.

The main goal of the system is the automation of the process of document creation. In the current state of the system a lot of manual work is required on the user's side. Every instance that is created needs to be enriched with semantic meaning by the user, either by annotation or by placing the instance in a proper context. Tangible benefits of usage are expected only when documenting the security properties of large applications and in cases the users are not experts and need constant guidance. All available options to handle more than one structural entity at a time should be estimated and incorporated into the tool.

A number of improvements in the state of the software system can be performed as future work. First of all, the Output Converter, Evaluator Support and Protection Profile integration modules have been proposed but need to be implemented and tested into practice. Some features are still missing and should further supplement the existing software system. The change classification module currently implements only the propagation and no propagation required types of changes and needs to be extended to the other possible modification types.

We created a document template for writing the Security Target. Document templates can be in future created for the assurance documentation as well. They can be provided as per CC Component and will be dynamically created on the basis of the component structure.

8.2 Results

Evaluation of the CCWord system was performed over the data from the relatively small, self-contained security documentation related to QuickQuery database. Part of this documentation was presented as a running example in this thesis. The testing and experiments were targeted at building a comprehensive set of examples to demonstrate that all functionality supported by the system operate correctly.

The conclusion that can be drawn from our evaluation results is that efficient, automatized management of change in IT Security Documentation is possible, although the role of the human factor continues to be of primary importance. As the test were done over a small, artificially created example, we still cannot conclude if the presented solution would lead to clear performance improving benefits in the process of change management.

- For the purpose of this thesis, decomposition is performed manually and relies on the judgement of a human factor. Also element dependencies identification performed manually .. Using this example for industrial applications will require this decompositions are performed by experienced users, or using natural language processing techniques and then proofread by experienced users.

Bibliography

[Bia09] Andrzej Bialas. Ontology-based security problem definition and solution for the common criteria compliant development process. *Dependability of Computer Systems, International Conference on*, 0:3–10, 2009.

[CCT01] *CCToolbox Installation Guide*, March 2001. CTAN: `http://pagenotes.com/writings/ccToolbox6f/CC%20Toolbox%206/Data/Docs/installation_guide.htm`.

[Cha] S.K. Chang. Handbook of software engineering and knowledge engineering.

[Com02a] ISO/IEC Standard 15408. *Common Criteria for Information Technology Security Evaluation*, version 3.1 edition, 2002. `http://www.commoncriteriaportal.org`.

[Com02b] ISO/IEC Standard 15408. *Common Criteria Part 1*, version 3.1 edition, 2002. `http://www.commoncriteriaportal.org`.

[Com02c] ISO/IEC Standard 15408. *Common Criteria Part 2*, version 3.1 edition, 2002. `http://www.commoncriteriaportal.org`.

[Com02d] ISO/IEC Standard 15408. *Common Criteria Part 3*, version 3.1 edition, 2002. `http://www.commoncriteriaportal.org`.

[Com09] ISO/IEC Standard 15408. *Common Evaluation Methodology*, version 3.1 edition, July 2009. `http://www.commoncriteriaportal.org`.

[CRGmW96] Sudarshan S. Chawathe, Anand Rajaraman, Hector Garcia-molina, and Jennifer Widom. Change detection in hierarchically structured information. In *In Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 493–504, 1996.

[DM07] Mathieu d'Aquin Marta Sabou Diana Maynard, Wim Peters. Change management for metadata evolution. In *International Workshop on Ontology Dynamics (IWOD) at European Semantic Web Conference*, 2007.

[EFGW07] Andreas Ekelhart, Stefan Fenz, Gernot Goluch, and Edgar R. Weippl. Ontological mapping of common criteria's security assurance requirements. In Hein S. Venter, Mariki M. Eloff, Les Labuschagne, Jan H. P. Eloff, and Rossouw von Solms, editors, *SEC*, volume 232 of *IFIP*, pages 85–95. Springer, 2007.

[Fog99] Karl Franz Fogel. *Open Source Development with CVS*. Coriolis Group Books, Scottsdale, AZ, USA, 1999.

[GGST86] H N Gabow, Z Galil, T Spencer, and R E Tarjan. Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. *Combinatorica*, 6(2):109–122, 1986.

[HT06] Micky Krause Harold Tipton. *Information Security Management Handbook*, volume 2, chapter 25. CRC Press, 2006.

[Hut09] Dieter Hutter. Semantic management of heterogeneous documents. In *MICAI 2009: Advances in Artificial Intelligence*, pages 1–14. Springer, 2009.

[JW00] John Boone John Faust Jim Williams, Allen Basey. Common criteria profiling knowledge base user guide, March 2000.

[KKK04] Haeng-Kon Kim, Tai-Hoon Kim, and Jae-Sung Kim. Reliability assurance in development processes for toe on the common criteria. 2004.

[Lam83] Butler W. Lampson. Hints for computer system design. In *IEEE Software*, pages 33–48, 1983.

[LAS05] Yaozhong Liang, Harith Alani, and Nigel Shadbolt. Ontology change management in protege. In *In Procs of the 1st AKT Doctoral Symposium, Milton Keynes*, 2005.

[MCV04] Paul Mason, Ken Cosh, and Pulyamon Vihakapirom. On structuring formal, semi-formal and informal data to support traceability in systems engineering environments. In *CIKM '04: Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 642–651, New York, NY, USA, 2004. ACM.

[MPS07] Diana Maynard, Wim Peters, and Marta Sabou. Change management for metadata evolution, 2007.

[Mül06] Normen Müller. An Ontology-Driven Management of Change. In *Wissens- und Erfahrungsmanagement LWA (Lernen, Wissensentdeckung und Adaptivität) conference proceedings*, pages 186–193, 2006.

[Mül07] Normen Müller. Towards an Ontology-Driven Management of Change. Exposé of PhD research proposal, June 2007.

[Nag05] William Nagel. *Subversion Version Control: Using the Subversion Version Control System in Development Projects*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.

[Nyg07] T. B. Nygaard. Common criteria design toolbox. Master's thesis, Informatics and Mathematical Modelling, Technical University of Denmark, DTU, Richard Petersens Plads, Building 321, DK-2800 Kgs. Lyngby, 2007. Supervised by Professor Robin Sharp and Assoc. Professor Michael R. Hansen, IMM DTU.

[RJ99] Bala Ramesh and Matthias Jarke. Towards reference models for requirements traceability, 1999.

[Sha06] R. Sharp. A SUMO-based domain ontology for the common criteria, oct 2006.

[SNH07] Arash Shaban-Nejad and Volker Haarslev. Towards a framework for requirement change management in healthcare software applications. In *OOPSLA Companion*, pages 807–808, 2007.

[tel] Telelogic. White paper. Available online (12 pages).

[tel01] *Telelogic - DOORS Ref. Manual: v5.1*, 2001.

[Tic85] Walter F. Tichy. Rcs a system for version control. In *Software: Practice and Experience*, pages 637–654. Copyright 1985 John Wiley and Sons, Ltd, Department of Computer Sciences, Purdue University, West Lafayette, Indiana 47907, U.S.A. DOI: 10.1002/spe.4380150703, 1985.

[wik] wikipedia. Common criteria wikipedia. `http://en.wikipedia.org/wiki/Common_Criteria`.

,

Glossary

Action	Evaluator action element of the CC Part 3, 15
Activity	Application of an assurance class of the CC Part 3, 15
CC	Common Criteria for IT Security Evaluation, 1
CEM	Common Evaluation Methodology, 15
Class	A group of components with a common topic., 14
Component	Smallest selectable units used in Security Targets to formalize the functional requirements or to specify the rigour or depth of evaluation., 14
EAL	Evaluation Assurance Level, 14
Element	A lowest level expression of a security need, 14
Family	A group of components that share a more specific focus., 14
Subactivity	Application of an assurance component of the CC Part 3, 15
Work unit	Most granular level of evaluation work., 16