Common Criteria Design Toolbox

Tore Bredal Nygaard

Kongens Lyngby 2007
IMM-MSC-2007-30

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

Abstract

The Common Criteria framework was developed by different organizations to create a common standard for improving comparability between the security of IT products. The framework involves creating *Protection Profiles* (PP) which are implementation independent security documents, describing security requirements for a family of IT products. Each of these PPs follow a defined structure that can be evaluated by PP evaluators.

This thesis presents the design and implementation of a prototype of a Toolbox which purpose is to help PP developers develop PPs, following this uniform structure.

Keywords: Common Criteria, Protection Profile, Security Target, Design, Toolbox, TOE

Resumé

Common Criteria frameworket blev udviklet af forskellige organisationer for at lave en fælles standard, der kan forbedre sammenligningen mellem sikkerheden af IT produkter. Dette framework involverer oprettelsen af *Protection Profiles* (PP), disse er implementations uafhængige sikkerheds dokumenter der beskriver sikkerhedskrav for en familie af IT produkter. Hver af disse PPer følger en defineret struktur der kan blive evalueret af PP evaluatorer.

Denne afhandling præsenterer designet og implementeringen af en prototype af en værktøjskasse, hvis mål det er at hjælpe PP udviklere med at udvikle PPer der følger denne ensartede struktur.

Nøgleord: Common Criteria, Protection Profile, Security Target, Design, værktøjskasse, TOE

Preface

This thesis was prepared at Informatics and Mathematical Modelling at the Technical University of Denmark in partial fulfillment of the requirements for acquiring a M.Sc. thesis in Engineering.

The project was developed in the period from 1st of September 2006 to the 2nd of April 2007 under the supervision of Robin Sharp and Michael R. Hansen.

Lyngby, April 2007

Tore Bredal Nygaard

Acknowledgements

I would like to thank my two supervisors, Robin Sharp and Michael R. Hansen, I would not have been able to make this possible without their help and guidance.

I would also like to thank all that contributed with mental support in times everything felt overwhelming and my wife for her understanding and support throughout the project. I would also like to thank the other students writing their theses at IMM at the same time as me for the discussions on various relevant and irrelevant topics.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this chapter the reason for writing this thesis will be explained. First the motivation behind the project will be presented, this is followed by a short description of the background behind it. After the background description this chapter describes the objective of what should be accomplished by this thesis. Finally a short description of the chapters of the report will be presented.

1.1 Motivation

In the last years a lot of focus has been put on having secure IT products. But what does it really mean to have a secure product? Currently it is a quite difficult task to verify which of two products, that are the most secure. Therefore it is important to use one standard that can be used by all product developers to evaluate their products. This enables a comparison between different IT products on equal terms in regard to security.

The development of security specifications can be a tedious and time consuming process. Because of this it is important that the security specification developers have a tool that ease the development of these specifications. The use of such tool can also ensure that all specifications follow the same guidelines and structure.

1.2 Background

Standards for defining secure systems have been developed by different institutes to provide means for evaluating different sets of software or hardware. Some years ago three standards merged into one called the Common Criteria, this was done to get a mutual base for evaluating different IT products, up against each other. One of the main ideas behind the Common Criteria is that it must reach a wide range of users. This is in line with the motivation that a common goal for security evaluation of IT products is needed. One of the problems with the Common Criteria is that security policies developed with them can be difficult to evaluate due to the structure, which the developers of them follow, can be different. The basic structure of the security policies is essentially the same in all policies being constructed by the Common Criteria, but the organizations that use them often have their own way of representing them.

The Common Criteria Toolbox created by SPARTA[1] gave developers a tool that could be used for starting the development of security policies following the Common Criteria. The aim of the CC Toolbox by SPARTA was partly to help developers define skeletons of Security Targets for the products they offer and partly to start the definition of Protection Profiles. According to SPARTA's web site, [SPARTA, 2007], their tool helped developers getting started by guiding the users through interview based decisions. Sadly it was discontinued in the beginning of 2004, this might be due to the problem that the output of the Common Criteria Toolbox developed was a HTML file that, if changed, would disallow for further development within the tool.

1.3 Objective

This project seeks to give developers a tool with which the process of making comparable security specifications will be made less time consuming. As described earlier the process of building security specifications is tedious and time consuming. The idea of the tool developed in relation to this thesis is to ease these bookkeeping processes while presenting the user with different relevant choices. The tool must also give the user means for sharing the outputs from the program with other users. In the remainder of this report the tool will be referred to as the *Common Criteria Design Toolbox* (CCDT).

[1] SPARTA was founded in 1979 as a system engineering and advanced technologies company.

1.4 Thesis Overview

The thesis is divided into the following chapters:

Chapter 1 Introduction: Presents the idea behind the thesis.

Chapter 2 Common Criteria: Provides information about the Common Criteria. This includes a definition of the terms used by the Common Criteria as well as providing a case study for use throughout the report.

Chapter 3 Problem Analysis: In this chapter decisions about the scope of this project are presented.

Chapter 4 Requirements Specification: Outlines the requirements to the tool developed. The requirements are based on a presentation of Use Cases that shows functionality that must be provided by the toolbox.

Chapter 5 Design & Implementation: Presents the decisions taken in the process of designing and implementing the tool.

Chapter 6 Discussion: A discussion of the problems and needs associated with the Common Criteria and the toolbox, as well as the functionality and usability of the two.

Chapter 7 Conclusion: A short conclusion defining what has been achieved by this thesis.

Chapter 2

Common Criteria

This chapter gives a presentation of the content of the *Common Criteria* (CC). The history of the CC will be presented, followed by a more thorough explanation of the concepts within the CC. The chapter ends with a case study describing how the CC are applied to a small example.

The information provided in this chapter comes from various sources, the main source is the three parts presented in [Criteria, 2006], references to these parts will in the following be written as CC part x, with x being either 1, 2 or 3. Also the description of the toolbox developed by SPARTA, [SPARTA, 2007], and the web site for the Common Criteria, [Portal, 2007], were used for background information on the Common Criteria. Furthermore [Pfleeger and Pfleeger, 2003] was used for background information about Computer Security.

The CC operate with the term *Target of Evaluation* (TOE) which will be used to refer to what is to be evaluated, that being a software, firmware or hardware product. The term *TOE Security Functions* (TSF), which is a set of the hardware, software and firmware of the TOE that must be relied upon, will also be used.

2.1 Introduction

The Common Criteria were developed as a union of existing standards to work towards a common platform for security evaluating products. They originated from the European standard ITSEC, the Canadian standard CTCPEC and the United States Department of Defense Standard, TCSEC. The Common Criteria are being developed by organizations from the following countries: Australia, Canada, France, Germany, Japan, Netherland, Spain, UK and the USA. The CC has been accepted as ISO/IEC 15408 as evaluation criteria for IT security.

The CC have three different classes of users, these being:

- Consumers - are using the CC to view results of evaluation as well as making Protection Profiles with the needed security requirements of their organization,
- Developers - are constructing profiles that satisfy the needs of the consumers and
- Evaluators - evaluate that the profiles made by the developers suit the specified needs.

The CC are to be used as a guide for development, evaluation and securing of IT products with security functionality. And as such it gives a basis for evaluation that permits comparability between different and independent products. It is important to note that the evaluation only has a meaning in its context and therefore any evaluation must be compared up against the needs of the product it must support.

The Common Criteria exist of multiple parts, one for defining how the security within the TOE must work, and another that defines how to evaluate that the TOE works as intended.

The Common Criteria are not by default limited to the three most commonly known computer security aspects, confidentiality, integrity and availability, as described in [Pfleeger and Pfleeger, 2003], but aspects not covered by these three can be included by the applications of the CC.

Furthermore the Common Criteria deal with three scopes. One is to define a general *Protection Profile* (PP) that can be applied to a wide range of IT product types. The other scope is to define a *Security Target* (ST) that takes an existing Protection Profile and tailors it to an implementation dependent specific

identified product. The final scope is to create an implementation representation where schematics for the actual product is provided.

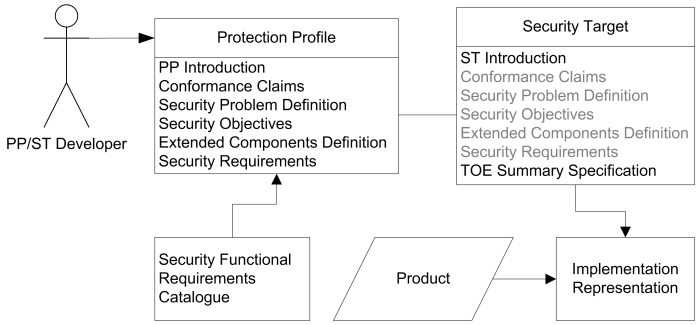

Figure 2.1: Common Criteria Overview

Figure 2.1 shows an overview of how the Common Criteria is structured. The PP/ST developer first creates an abstract PP that defines the security bounds in which any product type following this PP must operate. The PP uses components defined by the Common Criteria Part 2 to build its security Requirements. When the PP has been defined it can be used to specify security needs for a more clearly defined product. This builds the ST. Finally the ST can be implemented together with a product into an Implementation Representation defining the program design of the product. The items written in gray are items that are inherited from the PP. More on each part of the PP and ST will be presented in the following sections.

2.2 Protection Profile (PP)

The first of the Common Criteria schemes is a general one providing information about what a family of products must comply with.

According to CC Part 1 a *Protection Profile* (PP) is "*an implementation-independent statement of security needs for a TOE type.*"

The idea of the PP is to make a specification that can be used by various software or hardware developers.

The Protection Profile is divided into 6 sections, these being:

- PP Introduction

- Conformance Claims
- Security Problem Definition (SPD)
- Security Objectives (SO)
- Extended Components Definition
- Security Requirements (SR)

The PP Introduction contains information about the usage and major security features of the TOE as well as a PP reference that uniquely identifies the PP. Furthermore the PP introduction must identify what line of components the TOE belongs to. That could e.g. be antivirus programs or VPN[1] devices.

The Conformance Claims section presents claims about what version of the CC the PP follows.

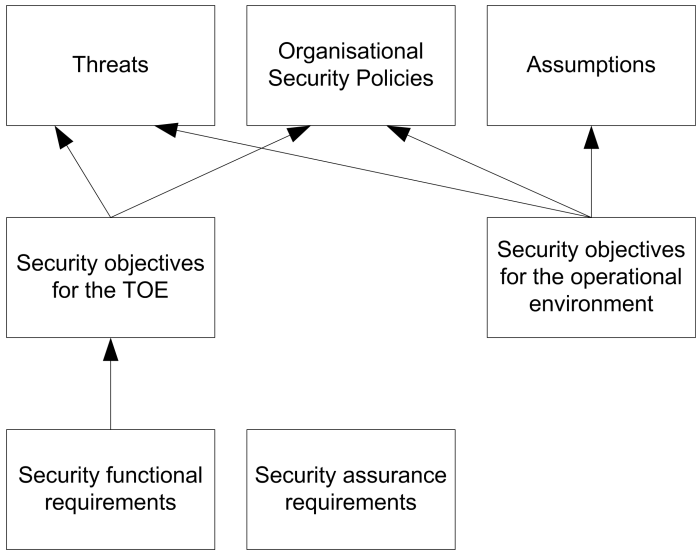

Figure 2.2: Relations between the security problem definition, the security objectives and the security requirements

Figure 2.2 is taken from [Criteria, 2006] Part 1 page 62. It shows the relationship between the Security Problem Definition, the Security Objectives and the Security Requirements. The following will elaborate on each of the areas.

[1] VPN: Virtual Private Network

The *Security Problem Definition* (SPD) section contains an identification of all threats to the system, assumptions about the working environment and all *Organizational Security Policies* (OSP) that exist in regard to the TOE. The threats must be defined in terms of a threat agent, e.g. "a hacker", an asset, e.g. "the TOE" and an action, e.g. "an intrusion".

As seen on Figure 2.2 on the facing page the *Security Objectives* (SO) addresses each defined Threat, Assumption and OSP. A Security Objective can either be an objective on the TOE or on the working environment - a TOE objective could be that the TOE must scan for viruses on a regular basis whereas an Environmental Objective could be that nobody can access the TOE without proper clearance. For each link that is created between a SPD item and a SO item, a rationale describing why the SPD is covered by the SO, must be provided.

The section with Extended Components Definition holds information about security requirements that are not defined by the Common Criteria. This is used when the CC does not specify suitable components for the TOE.

The *Security Requirements* (SR) are selected from the set of security and assurance components provided by the CC part 2 and 3. The structure and the scope of these will be presented in Section 2.4 on the next page. For each item defined in the Security Objectives at least one of the Security Functional Requirements from the CC part 2 has to be selected, unless the SO item is already covered by a component from the Extended Components Definition. Furthermore the Security Requirements must hold a definition of all terms used within the SFR components. The Security Functional Requirements often require the PP/ST developer to do decisions on the specific application of the SFR components. SFR components can be left incomplete to allow implementations of the PP to tailor each to their needs.

2.3 Security Target (ST)

A *Security Target* (ST) is according to CC Part 1 "*an implementation-dependent statement of security needs for a specific identified TOE.*". The ST follows the same structure as the PP, in this section only the parts of the ST not covered by the previous section will be explained.

In addition to the parts that exist in the PP, the ST holds an ST introduction that specifies which PP it implements as well as the scope for it. It also holds a TOE Summary Specification that identifies the TOE being implemented.

The Security Requirements section of the ST must ensure that each of the SFRs has been completed. For instance it is not allowed to specify a list of choices that the SFR can fulfill, it has to be narrowed down to a list that it must fulfill.

2.4 Security Functional Requirements (SFR)

This section presents the Security Functional Requirements as defined by CC part 2. The SFRs are divided into classes, each class contains a number of families. Each family contains a number of components and each component contains elements.

Each class short name starts with a F to identify that it is a Functional Requirement, the F is followed by a two lettered code that is taken from the name of the class. I.e. The class Security Audit will be named FAU.

In the Common Criteria version 3.1 there exist 11 classes:

- Security audit (FAU)
- Communication (FCO)
- Cryptographic support (FCS)
- User data protection (FDP)
- Identification and authentication (FIA)
- Security management (FMT)
- Privacy (FPR)
- Protection of the TSF (FPT)
- Resource utilization (FRU)
- TOE access (FTA)
- Trusted path/channels (FTP)

Each class contains a class introduction that describes the scope of the class and between two and 14 families.

A snippet from the class introduction of class FIA is:

> *Families in this class address the requirements for functions to establish and verify a claimed user identity.*

Within the class FIA the naming of the families consist of the short name of the class followed by a "_" and a three lettered code, for the family "User Authentication" the family short name is FIA_UAU.

A family consists of a family behavior describing the application of the family and a number of components. An example of such a behavior is for the family FIA_UAU within the FIA class:

> *This family defines the types of user authentication mechanisms supported by the TSF. This family also defines the required attributes on which the user authentication mechanisms must be based.*

And within the family the following components are found:

- Timing of authentication (FIA_UAU.1)
- User authentication before any action (FIA_UAU.2)
- Unforgeable authentication (FIA_UAU.3)
- Single-use authentication mechanisms (FIA_UAU.4)
- Multiple authentication mechanisms (FIA_UAU.5)
- Re-authenticating (FIA_UAU.6)
- Protected authentication feedback (FIA_UAU.7)

The components all hold information about how they relate hierarchically to each other, if a component relates hierarchically to another it means that it offers more security than the one it is hierarchical to.

> E.g. FAU_SAA.2 is hierarchical to FAU_SAA.1 so if FAU_SAA.1 was required to be used it is possible to use FAU_SAA.2 instead.

Components often have other components as dependencies, if this is the case the PP/ST developer has to include the depended component or a component that is hierarchical to the dependency. A component also holds information about when it applies, for instance the leveling information about FIA_UAU.2 states:

> *User authentication before any action (*FIA_UAU.2*), requires that users are authenticated before any other action will be allowed by the TSF.*

Furthermore the components holds tips in regard to what components from the *Security Management* (FMT) class, as well as information about which audit steps from the audit *Security Audit* (FAU) class, should be considered in regard to the component in question.

Each component also hold information about which elements it contains.

An example of an element is the element 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.*

Other elements require that the PP/ST developer fills out "blank" spots by using the four SFR operations. As defined in the CC part 1 pages 77-80 the four operations are:

- Iteration - "*Allows a component to be used more than once with varying operations*"
- Assignment - "*Allows the specification of parameters*"
- Selection - "*Allows the specification of one or more items from a list*"
- Refinement - "*Allows the addition of details*"

2.5 Security Assurance Components (SAR)

These are similar to the Security Functional Requirements, but provides means for assuring that the security of the TOE is investigated. The SARs are used as grounds for confidence that a TOE meets the SFRs defined in its PP or ST.

The SARs have a structure similar to that of the SFRs. They are also grouped into classes with subgroups of families. The SAR catalogue is presented in the CC part 3 and in this there exist various different classes for assuring that a TOE follows the security that it is defined to.

The SARs are used for obtaining an Evaluation Assurance Level, EAL. The point of the EAL is to make it possible to evaluate two products on more levels than just what Security Functional Requirements they follow.

There is a fixed list stating which SARs must be added to the Security Requirements for it to achieve a specific EAL. But in addition to those needed to achieve a specific EAL, there exist additional SARs that can be used for adding security choices to the PP or ST.

If two different products fulfil the same essential Security Functional Requirements then the one with the highest Evaluation Assurance Level is considered to be the most secure. This flexibility that the two axis evaluation gives, i.e. the evaluation of SFR and of the EAL, makes it possible for consumers to decide what particular product suits the needs of the organization best.

2.6 Case Study

This section holds a formalization of a Case Study. The formalization will start by defining the Security Problem Definition to some arbitrary device and then be extended to specifying the objectives taken to cover the it. This case study will be looked at in later chapters to provide the reader with means of comprehending the development of Protection Profiles.

2.6.1 Protection Profile Definition

The scope of this Protection Profile is to specify a secure environment for a TOE that is to be used by multiple users. The TOE in this example can be considered as a box that is connected to a network. This case study concentrates on a limited subset of the parts of the PP, these being the Security Problem Definition, the Security Objectives and the Security Functional Requirements.

The PP will be kept on a simple basis to ensure that it can be comprehended by the reader.

The Security Problem Definition within this PP can be presented as it is in Table 2.1.

Name	Definition
Security Problem Definition	
T.Unintended-Access	A user may gain unintended access to the TOE
T.Virus	A malicious agent may attempt to introduce a virus to the TOE
A.NoEvil	It is assumed that the administrators of the TOE do not deliberately cause any evil
A.Physical	The TOE is assumed to be placed somewhere that requires identification to enter
P.Antivirus-Definitions	It is dictated by the organization that antivirus definitions must be updated on a regular basis

Table 2.1: Security Problem Definition

The corresponding Security Objectives table is presented on Table 2.2. The PP author must define these Security Objectives by looking at what means are needed to counter the specified SPD.

Name	Definition
Security Objectives	
O.AntivirusUpdate	The TOE will update Antivirus definitions on a regular basis
O.Virus	The TOE will detect and take adequate actions against viruses
O.TOEaccess	The TOE must provide means of how it can be identified who accesses the TOE
OE.Physical	The IT environment must ensure that nobody can physically access the TOE without proper clearance
OE.NoEvil	The IT environment where the TOE acts shall ensure that the administrators are non-hostile

Table 2.2: Security Objectives

Each of the issues presented on Table 2.1 must be countered by at least one objective from Table 2.2. For each of the mappings between SPD items and Security Objectives, a rationale describing why the particular objective covers the SPD items is defined. For this case study it can be said that T.Virus is partially covered by O.AntivirusUpdate with the rationale that up-to-date antivirus definitions is a requirement for finding any viruses. It is also partially covered by O.Virus with the rationale that means for detecting viruses is needed to prevent viruses from doing harm.

The coverage of the SO over the SPD can be added to a coverage matrix which helps to illustrate that all SPD's has been covered. Figure 2.3 shows the coverage matrix to the case study.

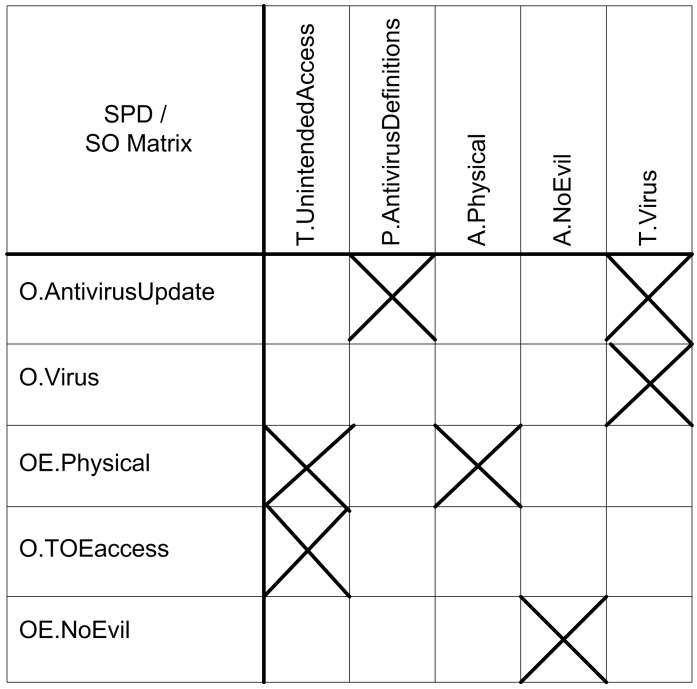

Figure 2.3: Coverage Matrix of SO covering SPD for Case Study

After all Security Objectives have been defined and linked to the corresponding SPD items each of these has to be linked to a Security Functional Requirement as defined in the CC part 2.

The link from SOs into SFRs is similar to the way the SOs were linked to the SPD items. In this example the FIA_UAU.2 component covers the O.TOEaccess since the user authentication provided by FIA_UAU.2 is an important part of the objective. FIA_UID.1 is a dependency to FIA_UAU.2 and it must therefore also be satisfied by the PP.

After each of the SOs has been mapped to corresponding SFRs, it is possible to present this as another coverage matrix like the one presented earlier. This matrix will not be presented here.

Adding a component to the Security Requirements is done by adding all elements it contains as well as all dependencies it must specify. After that it is up to the

PP/ST developer to either iterate, assign, select or refine the requirements.

2.7 Summary

This chapter presented the Common Criteria, including the relationships between the Protection Profile and the Security Target as well as the contents of the two. The chapter also included a presentation of the structure between classes, families, components and elements within the Security Functional Requirements.

The chapter was concluded with a Case Study that presented how a small example can be interpreted by the Common Criteria.

In the following chapter decisions on what topics from the Common Criteria should be implemented in the Common Criteria Design Toolbox will be presented.

Chapter 3

Problem Analysis

This chapter presents a presentation of and a discussion on the decisions taken in regard to the scope of this thesis. The chapter is divided into relevant sections for illustrating the overall topics. The sections below address overall topics from the development phase.

3.1 Consistency Checks

As defined by the Common Criteria there is some validation to be performed.

For a PP to be well formed it must consist of the six parts that were presented in Chapter 2 on page 5. The program must ensure that all parts are filled out with valid data. Within each part further checks must be performed, e.g. it must be verified that the PP introduction holds a PP reference and a TOE overview and that an assumption is never covered by a TOE Objective. It must also be checked that all selected Security Functional Requirements indeed exist in the Catalogue as well as no Security Objectives on the IT environment is covered by SFR components.

3.2 External Relations

The PPs and STs developed by the toolbox must be stored in a format that is widely known, commonly used and that can be shared between different platforms and programs. This is due to the main idea of the Common Criteria, that it must be possible for them to reach a wide audience. Since the document about the Common Criteria was provided as an XML[1] document it was decided to use XML as the output format from the program. Using XML as output format gives a wide range of application possibilities. An XML Schema, XSD, can be defined to ensure that the format of the PP/STs follow the required structure. And it is possible to define XML style sheets, XSLT, that transforms the output from the program into relevant information. For some systems this could be relevant for a subset of the PP/ST defined by the program, but it could also be a stylesheet for transforming the PP/ST into a format used for presenting the PP/ST as complete.

The CC part 2 defines SFR components that will be loaded into a catalogue that can be used for performing searches. The mechanism for loading the components into the system should be developed by looking at the structure of the SFR components as provided by the CC as XML.

3.3 Restrictions

It was decided to focus on a subset of the sections of the PP since various of these can be represented as mere text. The sections that should be implemented as more than just a textual representation are the **Security Problem Definition**, the **Security Objectives** and the **Security Requirements**. Furthermore only the Security Functional Requirements part of the Security Requirements will be implemented, this means that the program will not allow users to add Security Assurance Components, and consequently that no EAL assignment can be made.

One part that initially was considered out of the scope of this project is the part with the Extended Components Definition, since this is left out in this thesis it is not possible for PP/ST developers to use any components that is not provided by the CC part 2 within the program. However this part is too important to leave out completely, so the Chapter 5 on page 35 will present basic design decisions about how to integrate it. The way to define the Extended Components Definition is a project of its own and should be investigated.

[1] XML: Extensible Markup Language

Because of the above restriction to the Security Requirements section it was decided not to offer support for browsing the Security Assurance Components within the developed Catalogue.

Future versions of the toolbox should be extended to hold more complex implementations of the limited sections, or as minimum give means for referring to locations where the parts not included can be found.

3.4 Summary

This chapter gave an overview of what was to be focused on in regard to the development of the Common Criteria Design Toolbox. The need for a way to share the output from the program was identified and it was decided to handle this with XML. This chapter also outlined some basic restrictions to the program, one of the biggest was the way of defining extended components within the program. It was also decided to only deal with the aspect of the Security Functional Requirements.

The following chapter will specify what should be accessible for a PP/ST developer within the developed program.

Chapter 4

Requirements Specification

This chapter presents the Requirements Specification to the Common Criteria Design Toolbox developed in this thesis. Since the user of the program, the Primary Actor, is typically a PP/ST developer the program must support the requirements of such.

The intention of this program is to give PP/ST developers a tool that helps in developing PPs and STs for their IT products. That could either be for the PP developer to specify requirements for a TOE specifying a family of products or for the ST developer to identify this TOE further within the requirements.

The requirement specification is built up as Use Cases. This is done to illustrate what a user of the program should be able to expect. The Common Criteria Design Toolbox must be structured so the Use Cases presented below are made possible to access by the user.

The idea behind the structure of the Use Cases is as suggested by Alistair Cockburn [Cockburn, 2007].

The program should only allow one user at a time and should be independent on other processes, therefore only one actor will be used in the following.

In connection to the Use Cases the Case Study from Section 2.6 on page 13 is

being used. This is done to illustrate how and where the different parts of a PP/ST is added.

4.1 Use Cases Overview

This section holds an overview presenting how the use cases are structured as well as how they relate to each other.

The structure of the Use Cases is explained on Table 4.1.

UC.0 Use Case Format	
Goal	What will be accomplished upon success of the Use Case
Level	One of two: User: Actions carried out by the user Sub-function: Actions done by the system If nothing else is specified the Level is User
Preconditions	These conditions must hold before the steps in the Use Case can be performed
Success conditions	Describes a successful termination to the Use Case
Failure conditions	Describes a failed termination to the Use Case
Steps	
Step 1	Shows the steps needed to follow to get to the goal
Variations	
Step 1a	Shows any variations that might be to the point in the named step
Step 1b	There can be multiple variations to each step
Exceptions	
Name	If a Use Case yields wrong results, the step that can go wrong as well as a description of what could be wrong is written here
Return step	And the step to proceed from in case of an exception is labeled here

Table 4.1: Use Case 0: Use Case Format

On Table 4.2 a brief overview of the Use Cases is given.

Use Case Overview	
UC.1 New PP	Creates a new PP
UC.2 Alter PP	Alters the different sections of a PP
UC.3 Alter SPD	Specifies the Security Problem Definition by adding and/or modifying the Threats, Assumptions and Organisational Policies in regard of the system
UC.4 Alter SO	Specifies the Security Objectives by adding and/or modifying the TOE and Environmental Objectives
UC.5 Alter SFR	Specifies how the SFR Components for the Security Requirements are added
UC.6 Find Component	Locates SFR components in the Catalogue for use with UC.5
UC.7 Validate PP	Validates a PP internally in the system
UC.8 Import PP	Imports an existing PP
UC.9 New ST	Creates a new ST
UC.10 Import ST	Imports an existing ST
UC.11 Validate ST	Validates a ST internally in the system

Table 4.2: Use Case Overview

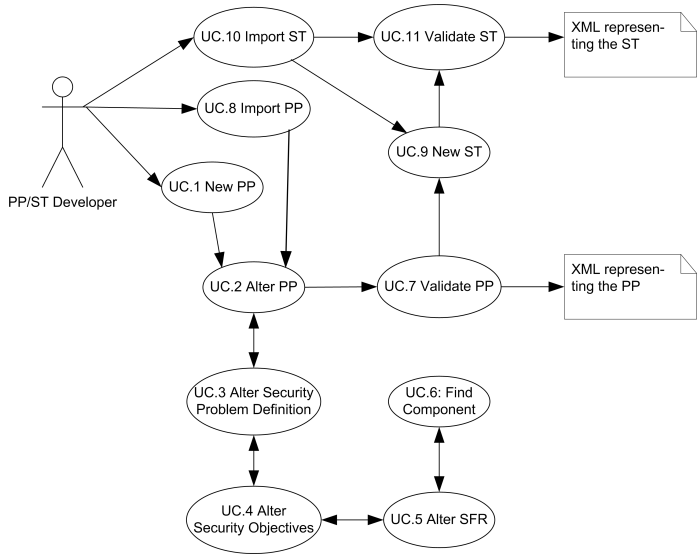

Figure 4.1: Overview of the connection between the Use Cases

Figure 4.1 shows how the Use Cases are connected as well as their connections to the outside.

The Use Cases does not define how the ST is altered, this is intended to be in a similar manner as the way the PP is altered, but with the limitations that are on STs.

Some of the steps in the use cases are merely steps that require the user to enter simple data. If this is the case no further description of the step is presented. More complicated steps will be presented as an additional use case.

To better comprehend the use cases a small example will be presented in parallel to these. The examples will be labeled for easier identification, each example will be labeled as follows: "Ex.[Relevant Use Case][Variation]", i.e. an example supporting Use Case 1 will be named "Ex.1".

4.2 Protection Profile

In this section the Use Cases supporting the creation of Protection Profiles will be presented.

UC.1 New PP	
Goal	Create a new PP
Preconditions	A loaded Catalogue exists
Success conditions	A PP has been stored
Failure conditions	No PP has been stored
Steps	
Step 1	Create Protection Profile Skeleton
Step 2	UC.2 Alter PP
Step 3	Save PP
Variations	
Step 3a1	UC.7 Validate PP
Step 3a2	Export PP
Exceptions	
Name	The PP does not validate
Return step	Step 2

Table 4.3: Use Case 1: Create new PP

Table 4.3 illustrates that the user starts the program, creates a new Protection Profile, modifies the default data and stores it either as a working edition or as a validated exported version.

Creating a PP skeleton creates a skeleton that follows the structure constraint but does not hold any values.

> Ex.1a: The user, a PP/ST developer, wants to create a new Protection Profile. After starting the program he creates a new skeleton of a PP and begins filling out its sections.

> Ex.1b: After entering information to all sections of the PP the user saves it to the hard disk.

UC.2 Alter PP	
Goal	Alter the body of a Protection Profile
Preconditions	A PP skeleton and a loaded catalogue exist
Success conditions	All sections of the PP contain information
Failure conditions	Not all sections contain information
Steps	
Step 1	Alter PP introduction information
Step 2	Alter Conformance Claims
Step 3	UC.3 Alter Security Problem Definition
Step 4	UC.4 Alter Security Objectives
Step 5	Alter Extended Components Definition
Step 6	UC.5 Alter SFR
Exceptions	
Name	Each step is selfcontaining if any errors occur
Return step	The step causing the error must be repeated

Table 4.4: Use Case 2: Alter PP

Step 1, 2, 3 and 5 from Table 4.4 may be done in any order the user desires, the Use Case merely illustrates the most straightforward use. However, step 4 must follow step 3 and step 6 must follow step 4.

> Ex.2a: The user fills out the PP introduction with a reference to other PP's that it elaborates upon as well as an overview of the TOE.

> Ex.2b: The Conformance claims are entered as conformance to the used Common Criteria version. Which with the current version would be Common Criteria v. 3.1 Revision 1 Part 2

The step 5 of Table 4.4 on the previous page should have a Use Case defining how to develop new components that extend the repository of the CC part 2. This was left out in this thesis on purpose. This is done since it is not a trivial task to define new components that conform with the Common Criteria which was also argued in Section 3.3 on page 18.

The Security Problem Definition is modified by either adding new SPD items to the PP or by removing or modifying existing ones.

UC.3 Alter Security Problem Definition	
Goal	Specify the Security Problem Definition
Preconditions	A Security Problem Definition skeleton exists
Success conditions	At least one SPD item exists in the SPD
Failure conditions	No SPD items exist in the SPD
Steps	
Step 1	Add a SPD Item
Step 2	Repeat above step or variations until satisfied
Variations	
Step 1a	Modify a SPD Item
Step 1b	Remove a SPD Item
Exceptions	
Name	The SPD Item that is sought removed is linked to a SO Item
Return step	Step 1b

Table 4.5: Use Case 3: Alter Security Problem Definition

Ex.3a: The user adds the threat "T.UnintendedAccess" with the definition "A user may gain unintended access to the TOE" to the SPD. He continues to add the remaining Threats, Assumptions and Organizational Policies from Table 2.1 on page 13.

UC.4 Alter Security Objectives	
Goal	Specify the Security Objectives
Preconditions	A Security Objectives skeleton exists
Success conditions	At least one SO item exists in the SO
Failure conditions	No SO items exist in the SO
Steps	
Step 1	Add a SO Item

continued on next page

UC.4 Alter Security Objectives	
Step 2	Repeat above step or variations until satisfied
Variations	
Step 1a Step 1b	Modify a SO Item Remove a SO Item
Exceptions	
Name Return step	The SO Item that is sought removed is linked to a SFR Component Step 1b

Table 4.6: Use Case 4: Alter Security Objectives

Ex.4a: The user adds the Objective "O.TOEaccess" with the definition "The TOE must provide means of how it can be identified who accesses the TOE" to the Security Objectives. The user adds the remaining Objectives from Table 2.2 on page 14 and provides rationale similar to the one before.

Ex.4b: After adding the objectives the user links first "O.TOEaccess" to the threat "T.UnintendedAccess" by stating the rationale that "O.TOEaccess ensures that nobody can access the TOE without being identified". Links are added for the remaining objectives.

UC.5 Alter SFR	
Goal Preconditions Success conditions Failure conditions	Specify SFR Components for the Security Requirements A loaded Catalogue and a SFR skeleton exist One or more SFR's has been found in the catalogue and added to the SR section No SFR Components added
Steps	
Step 1 Step 2	UC.6: Find Component Add Component
Variations	
Step 1a Step 1b	Remove Component Alter Component
Exceptions	

continued on next page

UC.5 Alter SFR	
Name	No found Components
Return step	Step 1

Table 4.7: Use Case 5: Alter SFR

Ex.5a: The user locates a component that matches the Security Objective items that was added earlier. The component is then added to the Security Functional Requirements. This is repeated until all Security Objectives has been covered.

Ex.6a: Using the Find component feature of the Catalogue, the user enters the keyword "identify" to search for a component that matches "O.TOEaccess", he also chooses one of three search options, either exact, wild card or synonym search. The user performs the search and sees that "FIA_UID.2", "User identification before any action", matches the objective. The user then adds the SFR component and states the rationale that "FIA_UID.2" ensures that before a user can use the TOE, he must be identified.

It should not be possible to link Environmental Objectives to SFRs since this is not allowed by the CC.

After linking the SFRs to the Security Objectives it must be possible to apply the four Operations allowed by the CC specification, see Table 4.8.

Ex.5b: After adding components, the user modifies them. This is done by using one of the four operations specified in the CC part 1.

The usage of the operations is explained thoroughly in the CC Part 1 pages 77 to 80. The system should support all of these operations.

Name	Input	Output
IterateOperation	Component	Component-list
AssignmentOperation	Component	Component
SelectionOperation	Component	Component

continued on next page

continued from previous page

Name	Input	Output
RefinementOperation	Component	Component

Table 4.8: CC Operations

UC.6: Find Component	
Goal	To locate a SFR component in the Catalogue
Preconditions	A loaded Catalogue exist
Success conditions	The desired SFR component has been found
Failure conditions	The desired SFR component has not been found
Steps	
Step 1	Open the catalogue
Step 2	Enter search criteria and search option
Step 3	Pick component
Exceptions	
Name	The Component was not found
Return step	Step 2

Table 4.9: Use Case 6: Find Component

If the user does not find the component he is looking for, he can add a different abstraction level to the search option by broadening his search to include synonyms of the word as well as simple keyword matchings.

Alternatively to storing the PP, the user can decide to validate it and export it as a working PP. Doing so will make it possible to import it later on for elaborative work or for use as starting off an ST.

When validating the PP all sections must conform with the requirements provided by the CC.

In this thesis the focus has been put on validating the items presented on Table 4.10.

UC.7 Validate PP	
Goal	Validate a PP
Level	Sub-function
Preconditions	SPD, SO, SFR exist
Success conditions	All sections are valid

continued on next page

UC.7 Validate PP	
Failure conditions	At least one section does not comply with the check
Steps	
Step 1	Check SPD items
Step 2	Check SO items
Step 3	Check that all SPD's are covered by SO's
Step 4	Check SFR components
Step 5	Check that all SO's are covered by SFR's
Exceptions	
Name	A section does not validate
Return step	Exit with failure

Table 4.10: Use Case 7: Validate PP

The validations performed are:

- The SPD items must consist of a name and a definition.
- The SO items must consist of a name and a definition.
- Each SPD item must be covered by at least one SO item.
- The SFR components must exist in the SFR catalogue.
- Each SO item must be covered by at least one SFR component.

4.3 Security Target

After a PP has been created a user can initiate the development of a Security Target.

The development of a ST starts by importing a PP. After the PP to be imported has been selected it is validated by the system.

UC.8 Import PP	
Goal	Importing an existing PP
Preconditions	None
Success conditions	A valid PP is loaded into the system
Failure conditions	No PP is loaded
Steps	
Step 1	Select PP
Step 2	UC.7 Validate PP
Exceptions	
Name	The PP does not validate
Return step	Step 1

Table 4.11: Use Case 8: Import PP

In addition to importing the PP it is also possible to open it, the opening of a PP will not cause any validation and therefore it will not be possible to start a ST before the PP has been validated.

UC.9 New ST	
Goal	To create a new ST
Preconditions	A loaded Catalogue exist
Success conditions	A ST is stored
Failure conditions	A ST is not stored
Steps	
Step 1	UC.1 New PP
Step 2	Review and enter ST introduction information
Step 3	Review Conformance Claims
Step 4	Review Security Problem Definition
Step 5	Review Security Objectives
Step 6	Review Extended Components Definition
Step 7	UC.5 Alter SFR
Step 8	Save ST
Variations	
Step 1a	UC.10 Import ST
Step 1b	Load ST
Step 1c	UC.8 Import PP
Step 8a1	Validate ST
Step 8a2	Export ST
Exceptions	
continued on next page	

UC.9 New ST	
Name	Each step is selfcontaining if any errors occur
Return step	The step causing the error must be repeated

Table 4.12: Use Case 9: Create new ST

Ex.9a: The user selects the PP that holds the basis for the ST via the import mechanism Step 1c. This causes the PP to be validated using UC.7.

Ex.9b: The user reviews all sections and sees that all data matches the scope of the ST he is building.

Ex.9c: The user elaborates on the added components by modifying them with the use of the previously presented four operations.

The modification is done until all components are completed, meaning that no further information can be entered to each component.

Ex.9d: After all sections of the ST matches the desired, the user exports the ST.

UC.10 Import ST	
Goal	Importing an existing ST
Preconditions	None
Success conditions	A valid ST is loaded into the system
Failure conditions	No ST is loaded
Steps	
Step 1	Select ST
Step 2	UC.11 Validate ST
Exceptions	
Name	The ST does not validate
Return step	Step 1

Table 4.13: Use Case 10: Import ST

An alternative to importing a PP to use as base for an ST, it is possible to start the construction of a ST by importing a previously exported one. UC.10

illustrates this procedure. There has not been defined any use cases on how to alter the ST, this should be done in a similar way as that of the PP, but with the limitations that everything from the PP is locked, it should however be possible to apply the CC operations to the SFR components.

UC.11 Validate ST	
Goal	Validate a ST
Level	Sub-function
Preconditions	SPD, SO, SFR exist
Success conditions	All sections are valid
Failure conditions	At least one section does not comply with the check
Steps	
Step 1	Check SPD items
Step 2	Check SO items
Step 3	Check that all SPD's are covered by SO's
Step 4	Check SFR components
Step 5	Check that all SO's are covered by SFR's
Step 6	Check that all SFR's are completed
Exceptions	
Name	A section does not validate
Return step	Exit with failure

Table 4.14: Use Case 11: Validate ST

The validation of a ST is similar to the one of validating the PP, in addition to the validations done in the PP, it is necessary to verify that all SFR Components are completed.

4.4 Summary

This chapter illustrated what should be expected for PP/ST developers to do with the developed Common Criteria Design Toolbox. It used the approach of using Use Cases for presenting functionality. The basic needs of the program was that it should be possible to create a PP that could be exported to (or imported from) an XML format that could be used for sharing the PP with other tools using the same XML format. During the construction of the PP the developer should be able to get help with finding which SFR components that best suit the PP.

The following chapter provides decisions as well as implementation examples

from what was designed and implemented.

Chapter 5

Design & Implementation

This chapter presents the design and the implementation of the program developed in regard to this thesis work. The chapter is structured so that first the tools developed for supporting the main program will be discussed, thereafter decisions on the actual program will be presented, including how the different parts relate to the developed supporting tools. Furthermore in each section of this chapter first the design will be presented, this will be followed by the decisions on the implementation.

Within this chapter some code snippets made in SML[Hansen and Rischel, 1999] will be presented. These show how the various parts of the program is designed. Within the SML some references exist to the DTD provided by the Common Criteria. The DTD describing the *Security Functional Requirements* (SFR) can be found in Appendix A.1 on page 58.

The program is divided into two main components. The first is a catalogue used for looking up SFRs, the other is for designing the actual PP/ST. The PP/ST component uses the SFR catalogue in its development. These components will be described in the following.

5.1 Environmental Requirements

It was decided to develop the Common Criteria Design Toolbox in the Microsoft .NET Framework Version 2.0 [Microsoft, 2007]. This means that the Microsoft .NET Framework v. 2.0 must be installed on any system where the Common Criteria Design Toolbox is desired to run. Furthermore it is required that WordNet 2.1[WordNet, 2007] is installed due to parts of the Toolbox are using the dictionary functionality of this module. In addition to this it is also required that the XML holding the Common Criteria document is accessible.

5.2 SFR structure

This section shows how the SFRs provided by the CC part 2 can be structured in SML and how it is structured within the Toolbox. Figure 5.1 on the facing page and Figure 5.2 on page 38 hold the SML representation of the SFR structure.

The simple types from Figure 5.1 on the facing page hold a definition for the data type *level*, this is interpreted as information about what level any requirement is on, this being either a class, a family, a component or an element. The string holds its short name. In the implementation it was decided to make a method that returned what level a requirement was on, based on its name to reflect this data type. The regular expressions[REGEX, 2007] for the naming of the different levels are shown on Table 5.1.

Level	Regular Expression
Classes	F[a-zA-Z]{2}
Families	F[a-zA-Z]{2}_[a-zA-Z]{3}
Components	F[a-zA-Z]{2}_[a-zA-Z]{3}.[0-9]
Elements	F[a-zA-Z]{2}_[a-zA-Z]{3}.[0-9].[0-9]

Table 5.1: Regular expressions for the level of requirements

Figure 5.2 on page 38 shows how the SML data type level is used as a map from its name into its contents. The contents for each level are as defined earlier.

```
1  (* SFR requirements *)
2  datatype level = Class of string | Family of string |
3                   Component of string | Element of string;
4  (* Defines the different levels a requirement is defined on *)
5
6  type fcIntroduction = string;
7  type ffBehaviour = string;
8  type fcoHierarchical = string;
9  type fcoDependencies = string;
10 type fcoLevelling = string;
11 type fcoManagement = string;
12 type fcoAudit = string;
13
14 datatype foption =
15 s of string
16 | FE of foption list
17 | FEA of foption list
18 | FES of foption list;
19 (* Defines the different parts a SFR element holds as        *)
20 (* described in the dtd                                      *)
21 (* s corresponds to the CDATA                                *)
22 (* FE corresponds to the fe-item                             *)
23 (* FEA corresponds to the fe-assignment                      *)
24 (* FES corresponds to the fe-selection
```

Figure 5.1: SFR Type definitions

5.3 SFR Catalogue

The Catalogue provides means for searching for classes, families and components within the CC part 2. A search can be performed either by performing an exact, a wildcard or a synonym search, depending on the needs of the user. Furthermore the catalogue makes it possible to add SFRs to the PP being developed.

Figure 5.3 on the next page shows the types used for designing the Catalogue.

Figure 5.4 on page 39 shows the relationship of the different components within the Catalogue. Each of these components will be presented below.

```
type felement = (level, foption list) Polyhash.hash_table;
(* A map from a f-element name to its contents *)

type fcomponentMap = (
        level,
        fcoHierarchical * fcoDependencies * fcoLevelling *
        fcoManagement * fcoAudit * felement)
        Polyhash.hash_table;
(* A map from a f-component name to its content *)

type ffamilyMap = (
        level,
        ffBehaviour * fcomponentMap) Polyhash.hash_table;
(* A map from a f-family name to its content *)

type fclassMap = (
        level,
        fcIntroduction * ffamilyMap) Polyhash.hash_table;
(* A map from a f-class name to its content *)
```

Figure 5.2: SFR Mappings

```
(* Catalogue Types *)
datatype searchOption = Exact | Wildcard | Synonyms;
type keyword = string;
type index = (keyword, level list) Polyhash.hash_table;
```

Figure 5.3: Catalogue Simple Types

5.3.1 Parser

For making the catalogue it was necessary to parse the XML provided by the CC. The main idea is to structure the Common Criteria Security Functional Requirements into a list of classes, where each class contains information about its contents. The information from this parsing can then be formatted in different ways to present it as best seen fit in the situation.

5.3.1.1 Design

The parsing must be based on the nodes defined by the DTD structure Appendix A.1 on page 58 and the output should be on a structural representation

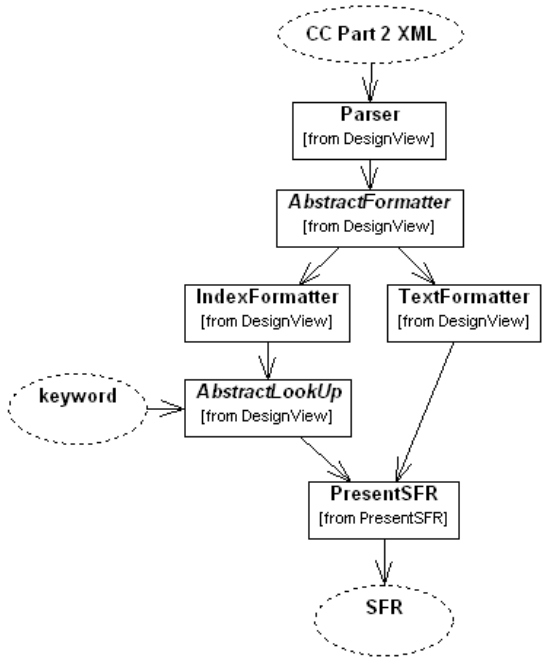

Figure 5.4: UML of the connections between the parsing, the formatting and the presentation of SFRs

similar to that of the SML representation presented above.

5.3.1.2 Implementation

The Parser collects information about all Security Functional Requirements and places it in the classMap as defined on Figure 5.2 on the facing page. In C# this is implemented as shown on Figure 5.5 on the next page.

The parsing is performed in C# by calling a recursive function, parse. The parse method is implemented as a switch case that hold all relevant XML tags and act upon the different tags. The parsing is performed both for generating the structure but also for entering the text that is needed to represent the different parts of the CC. This text is then stored within the structure so that the SFR structure presented above is preserved. The reason C# was chosen for

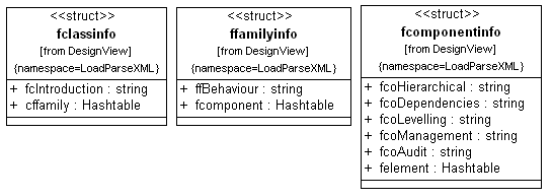

Figure 5.5: SFR structures in C#

performing the parsing was that it could then be done on the fly if the CC xml was updated.

5.3.2 Formatter

This part deals with formatting the data that was parsed using the parse methods. The idea of the formatters is to provide the user of the program with different means for formatting the CC part 2.

5.3.2.1 Design

The system should implement three different ways of formatting the parsed data, one way should be to make it possible to perform searches within the parsed data to identify relevant components for relevant Security Objectives. The other is meant for presenting the found areas in an easy to read manner. And finally the data should be formatted so it is possible to export the CC catalogue to SML for future consistency checks and well-formedness evaluations.

5.3.2.2 Implementation

The structure of the classes for performing the formatting is presented on Figure 5.6 on the facing page.

The C# library LoadParseXML.dll holds the classes and methods for performing the parsing. The DLL consists of a Parser class, an AbstractFormatter class and three generalizations of the Abstract Formatter. Each of the three generalizations represents a different way of presenting the SFRs, the scope of each of these is presented on Table 5.2 on the next page. The main idea is that different parts of the program must present the data provided by the CC differently.

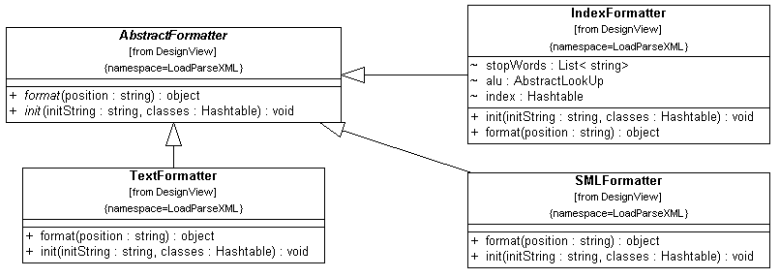

Figure 5.6: Formatting

Generalization	Scope
IndexFormatter	Returns a Hashtable holding the result of a gathering of all occurrences of all relevant words within the CC part 2 and stores them in an index for fast lookup.
TextFormatter	Returns a textual representation of the SFR at the given location as well as a textual representation of all sub elements to the looked up location.
SMLFormatter	Returns a SML representation of the SFR at the given location as well as a SML representation of all sub elements to the looked up location. The SML follows the structure defined on Figure 5.2 on page 38.

Table 5.2: The scope of the three Abstract Formatter generalizations

Each of the formatters is initialized with a Hashtable holding the data that was parsed by the Parser, as well as an initialization string, unique for each formatter. The initialization string for the Index Formatter holds information to be passed on to the Lookup mechanism. The initialization strings for both the SML and the Text formatters is currently not used, but enables developers to pass on lead-in information.

In addition to the initialization method they all hold a public method, format. In the Index Formatter, the format method returns the index holding all locations and occurrences of relevant words. The format method for the SMLformatter returns an SML representation of all classes within the parsed information and finally the TextFormatter returns either all classes represented as text or, if a specific location is given, text representing that particular position.

Some restrictions were put on the words used for making the index so ordinary words would not be added to it. This list of "Stopwords" includes common words as "to", "of" and "it". Furthermore it was decided that only words occurring on the Class, the Family and the Component level would be added. In addition to the skipping of simple words, the IndexFormatter stores the stem of the words being looked up, this is done so that later searches would be easier to perform, e.g. the formatter finds an occurrence of the word "securing", the stem of this word is "secure" which is the word being stored. Later when the user tries to perform a search, WordNet is again used to find the stem of the search keywords the user states, so if he tries to look up either "securing" or "secure" he will find this occurrence. In addition to storing the information about the context that the words were found, it is also stored how many occurrences within the context it has. Parts of the source code for the IndexFormatter is presented in Appendix A.2 on page 60.

An alternative to auto generating the index is, that it could be built up by security concepts defined by users. Due to the structure of the SFRs, that each component is placed within a family that again is placed within a class, this would not give as much value as the possibility of performing searches within the document.

The parsed information about the CC part 2, as well as the index, are both stored to a default folder and can be copied to this to avoid a large setup time. If this is not done upon the first search within the Toolbox, it will automatically scan through the CC.xml file and create a new parsed storage as well as a new index.

5.3.3 Lookup functionality

There is various ways for performing searches in systems, one way would be to scan the text for occurrences of words every time a search is initialized, this way was considered infeasible since each user of the system would have to search through the catalogue multiple times for each Security Objective that was sought covered. To circumvent this, it was decided to create an index holding all occurrences of the words used in the CC part 2 as well as the locations each word exists in.

5.3.3.1 Design

The structure of the index is presented on Figure 5.3 on page 38. And for this the lookup function

```
Lookup : keyword list * searchOption -> level list
```

exists.

5.3.3.2 Implementation

Figure 5.7 shows how it was made possible to use different approaches for looking up words within the index. One way, which is the one currently being used, is to use the WordNet library to perform lookups on the stem of the input keywords. Another way could be to gather a list of the most used words of a user, this approach has not been implemented and is merely presented as a possibility for future work.

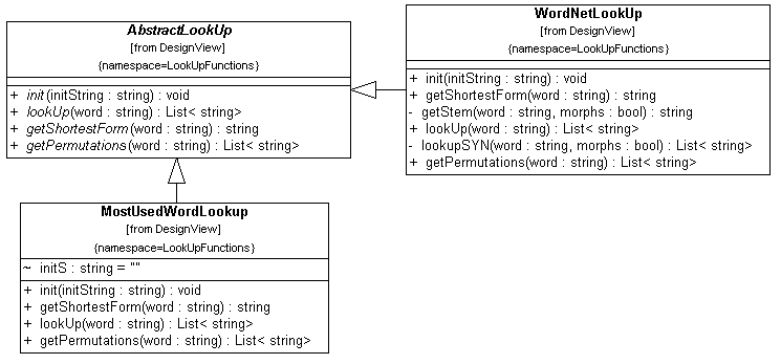

Figure 5.7: Lookup

The WordNetLookup approach enables the user to in addition to searching for the stem of the words he can search for a synonym of the word, so by searching for e.g. ”safe” he would be able to find the occurrence of ”securing” that was added earlier. The result for this search, as it is presented in the Toolbox, is shown on Figure 5.8 on the following page.

This implementation also holds means for searching for a wildcard within the Index, e.g. ”sec” would yield the same results as either ”securing” or ”secure”.

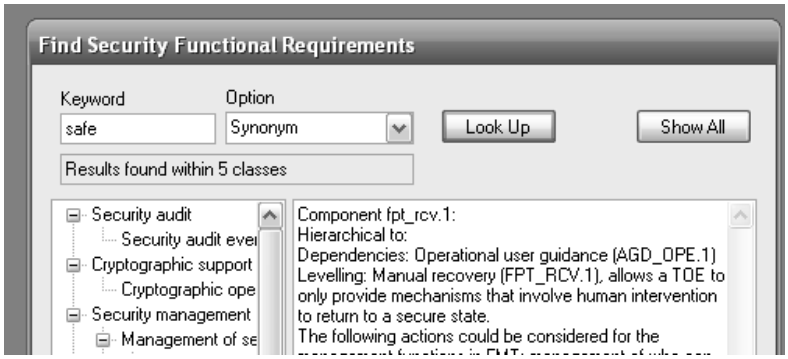

Figure 5.8: Synonym search for "safe"

With the current implementation it takes approximately 15 minutes to go through the parsed data gathering all relevant words and their locations. If the implementation would not have included the index created by the IndexFormatter from above, but real time lookup, these 15 minutes would have been an overhead that would have been carried out each time a lookup is performed. Lookups within the implemented Index, takes less than a second.

5.3.4 Present Functionality

After performing the lookup on relevant keywords, the found locations are passed on to the TextFormatter for making it possible to present the content of the found locations.

5.4 Protection Profile

The main purpose of the Toolbox being developed is to create a PP, the creation of a PP includes using the catalogue that was defined in the previous sections, furthermore it requires some functionality to connect the different sections of it together. This section will present how a PP is structured as well as how the implementation of it has been performed.

5.4.1 Design

Figure 5.9 presents how a Protection Profile is designed in SML. Line 36 shows what parts that must be constructed to encompass the various parts of the PP.

```
1  (* Protection Profile requirements                                    *)
2  type PPreference = string;
3  type TOEoverview = string;
4  type PPintroduction = PPreference * TOEoverview;
5
6  type CCconformanceClaim = string;
7  type PPclaim = string;
8  type ConformanceRationale = string;
9  type ConformanceStatement = string;
10 type ConformanceClaims = CCconformanceClaim * PPclaim *
11                         ConformanceRationale *
12                         ConformanceStatement;
13
14 datatype SPDname = T of string | P of string | A of string;
15 type SPDdefinition = (SPDname, string) Polyhash.hash_table;
16
17 datatype SOname = O of string | OE of string;
18 type SOdefinition = (SOname, string) Polyhash.hash_table;
19 (* Both the two datatypes SOname and SPDname are mappings from *)
20 (* a member in a disjoint set into a string defining it         *)
21
22 type Rationale = string;
23 type ExtendedComponentsDefinition = string;
24
25 type SORdefinition = ((SPDname * SOname), Rationale)
26                                          Polyhash.hash_table;
27 type SecurityObjectives = SOdefinition * SORdefinition;
28
29 type SRdefinition = fcomponentMap;
30 type SecurityRequirementsRationale = ((level * SOname),
31                                       Rationale)
32                                       Polyhash.hash_table;
33 type SecurityRequirements = SRdefinition *
34                             SecurityRequirementsRationale;
35
36 type ProtectionProfile = PPintroduction * ConformanceClaims *
37                          SPDdefinition * SecurityObjectives *
38                          ExtendedComponentsDefinition *
39                          SecurityRequirements;
```

Figure 5.9: SML representation of a PP

Creating a PP involves defining a Security Problem Definition by adding threats, assumptions and organizational policies to the SPDdefinition map. It also involves creating Security Objectives by adding TOE objectives and Environmental Objectives to the SOdefinition.

As indicated on Figure 5.9 a threat within a SPD is created as follows:

```
val spddef = mkPolyTable(10,error) : SPDdefinition;
val tSPDname = T "UnintendedAccess";
val tDefinition =
        "A user may gain unintended access to the TOE";
insert spddef (tSPDname,tDefinition);
```

The same goes for the other members of the SPDname datatype and similar for the SOname datatype.

When both the SPDdefinition and the SOdefinition have been constructed the different items in both of them are linked together. This is done as follows:

```
val Rationale =
        "Nobody can access the TOE without being identified";
insert sordef ((T "T.UnintendedAccess", O "O.TOEaccess"),
                                                Rationale);
```

5.4.2 Implementation

The example from the design section where a threat was added to the SPD of a PP happens similarly in the implementation. A threat in the implementation consist of a Prefix, namely the "T", a Name, the "UnintendedAccess" and a Definition, "A user may gain unintended access to the TOE".

To the user this is presented as shown on Figure 5.10.

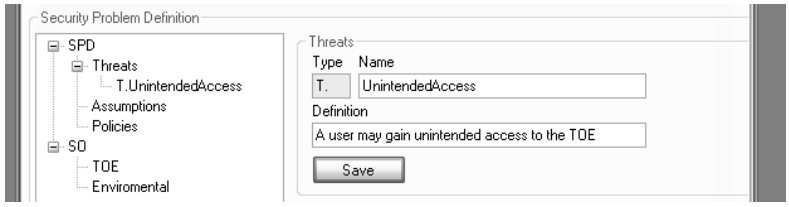

Figure 5.10: Adding a threat

When linking SOs to SPDs in the implementation an object, LinkConnected, consisting of the SPD, the SO and a rationale linking the two is created, this object is then added to a list containing all links within the PP.

The SML definition of a PP is in the implementation presented in two different ways, the GUI part is shown on Figure 5.11 on the next page and the structure behind is constructed so that everything the PP shares with the ST is defined in one class, ProfileSchemeContents, and both the PP and the ST extends this class. Therefore it can be said that the class ProtectionProfileContents Appendix A.4 on page 63 reflects the structure presented on Figure 5.9 on the previous page.

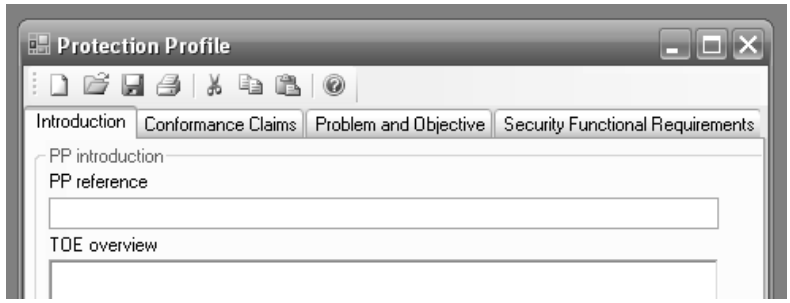

Figure 5.11: PP GUI representation

5.5 Common Criteria Design Toolbox

The Toolbox is built as a GUI on top of a class structure defining the basis of a PP and a ST. First a general scheme describing all commonalities between the PP and the ST were defined, this was followed by both a PP and a ST class that both implements the general scheme. By doing this it was easily possible to XML serialize the different classes into their corresponding XML documents. It is equally possible to de-serialize the XML files into the classes whereafter the GUI implementation could update the view from either the PP or the ST. Furthermore it makes it possible to import a PP into a ST by using the name of the PP as the PP Conformance in the ST and then using all the elements of the general scheme within the ST. The actual implementation in regard to implementing a PP into a ST was left out of the program due to time constraints.

If a stored version of the SFR catalogue does not exist the parser and formatter described before are initiated.

The base location for the XML document holding the Common Criteria SFRs can be changed in a settings file that is provided together with the executable.

Viewers can be implemented for presenting the XML, a prototype of such a viewer has been created in PHP[1]. It takes an PP on the XML form and transforms it with a XML stylesheet "XSLT" for showing the coverage matrix of the Security Objectives over the Security Problem Definition. The stylesheet is just made as a sample of what can be implemented. The output of the simple viewer is presented in Appendix A.6 on page 66. Besides this a stylesheet for transforming the XML into SML code was created, the resulting SML can be

[1]PHP: Hypertext Preprocessor

executed in extension to the SML that defined the structure of the PP as well as the SFRs. The resulting SML can be found in Appendix A.7 on page 67. This SML code was then executed after the definition of all the simple types as well as the SML functions defining the well-formedness of a Protection Profile. Appendix A.8 on page 69.

A snippet of the transformed output corresponding to the example from above is:

```
val fcompMap = Polyhash.mkPolyTable(10,error) : fcomponentMap;
val felementMap = Polyhash.mkPolyTable(10,error) : felement;
val secReqRat = Polyhash.mkPolyTable(10,error)
                        : SecurityRequirementsRationale;
```

and followed by

```
insert spddef (T "T.UnintendedAccess",
                "A user may gain unintended access to the TOE");
Polyhash.insert sodef (O "O.TOEaccess", "The TOE must provide
                means of how it can be identified who
                accesses the TOE");
Polyhash.insert sordef ((T "T.UnintendedAccess", O "O.TOEaccess")
                , "Nobody can access the TOE without being
                identified");
val secObj = (sodef, sordef);
Polyhash.insert felementMap (Element "fia_uid.2.1", [s " The TSF
                shall require each user to be successfully
                identified before allowing any other TSF-mediated
                actions on behalf of that user. "]);
Polyhash.insert fcompMap ( Component "fia_uid.2",( "FIA_UID.1", "",
                "User identification before any action
                (FIA_UID.2), requires that users identify
                themselves before any other action will be
                allowed by the TSF.", "the management of the
                user identities.", "", felementMap) );
Polyhash.insert secReqRat ((Component "fia_uid.2", O "O.TOEaccess"),
                "fia_uid.2 ensures that before a user can use the
                TOE he must be identified");
```

The snippet above shows how some items from the case study looks when the exported XML is transformed into SML. This snippet can be used for applying well-formedness functions to the Protection Profiles.

5.6 Verification

When testing IT software there are various ways of approaching it. One way is internal testing where every single part of the program is tested, if all these tests are completed with a successful result it is safe to argue that the program is working as intended. Another approach is external testing, where it is tested that all requirements are fulfilled. Besides these functional tests it is also important to test that the system can be used for the purpose it is intended, for instance to make user tests to verify that the program solves the problems described in its objective.

One of the strengths with the way the CC Design Toolbox was put together is that each of the crucial parts in it is module based. The CC Design Toolbox loads different libraries that all can be tested independently.

Every library that was used for creating this Toolbox was informally tested to verify that they gave the expected results. The WordNet library was tested in regard to the lookup of stems and synonyms, a set of words were looked up both in the WordNet application and in a test program that was created with the intention to informally test each module. The results of looking up the words in these two places yielded the same result, so it is assumed that the correctness of the implementation of it in regard to this program is just as correct as the one for the original program.

When testing the formatter focus was put on seing that the correct amount of classes, families, components and elements were shown. It was also verified that an arbitrary set of the components had the correct properties, i.e. correct leveling, hierarchical, dependency information. If the result of this test yielded a successful result, it is most likely that the parser providing the data also is functioning correctly.

The Toolbox as a whole was not tested, but it was seen that loading a PP, modifying it and exporting it again did not ruin the structure of it, nor did it change data that was not changed after the import.

5.7 Summary

This chapter presented the Design and Implementation of the Common Criteria Design Toolbox. The chapter showed the structure of the Toolbox and gave an overview of which components that exist within it. The chapter also showed the

decisions on how to verify that each developed part did what it was supposed to do.

The following chapter presents a discussion on the topics raised in relation to this thesis work, as well as a status on what has been accomplished.

Chapter 6

Discussion

This chapter presents a discussion on the issues dealt with in this thesis. The chapter will provide the status of the developed program as well as suggesting improvements to the program, this will be followed by a discussion on different approached on how to develop security documents. The chapter will also hold a general discussion on the problems found while writing the thesis and finally the program developed will be set in relation to the world it must act in.

The work behind this thesis, has resulted in a Toolbox that should be considered as a good first step towards the development of Protection Profiles. The program makes it possible to define a Security Problem Definition, the relevant Security Objectives to the Security Problem Definition. Similarly it enables finding Security Functional Requirements that suit the Security Objectives. Furthermore it can load the catalogue of Security Functional Requirements defined by CC part 2 into the system. When the PP has been created it will export the result to an XML file that can be plugged into different viewers or transformed into code for external validation.

However, due to timing constraints, the Toolbox is yet to be extended with a tool for applying the CC operations to the SFR components. This tool is required before the Toolbox should be considered a complete prototype.

6.1 Extending the Toolbox

Usually PPs are developed by more than one person, therefore a multiuser system should be integrated in the program. Multiuser support could be done with either CVS or Subversion, but it might be difficult to deal with, since the output is represented as XML, a solution could be to write a version controller that handles changes to different parts of the XML document. Otherwise it could be solved by storing each XML element in the documents as a document for itself and then keep track of which different parts exist in which PP. Another option would be to store the different parts of the Protection Profile or Security Target as mere text and save the XML transformation until they are considered complete. The latter version might introduce other structural difficulties, these will not be discussed here.

When PP/ST developers from an organization define new Security Problem Definition items and Security Objectives these could be stored as name and definition in a repository accessible by the entire organization. If this is done it might make it easier to construct future PPs within the same organization.

6.2 Development Integration

This section presents thoughts about the way the Common Criteria are used in relation to IT product development.

The creation with the Common Criteria themselves resembles a typical waterfall model since after a PP has been developed, and the development of a ST is started, it is not possible to change requirements stated within the PP without redesigning it all over again. The development of the PP can be seen as an incremental model since it is possible to redefine the Security Problem Definition after the work on Security Objectives and Security Requirements has started. Every change in the initial definition cause all dependent parts to be updated.

If the waterfall model was to be applied to the life cycle of an IT product being developed by using the Common Criteria, the CC part 2 would be the requirements phase, the specification, design and development phases would be covered by the CC part 3. The evaluation part of the CC would correspond to the test phase.

One advantage of the waterfall model in relation to the CC, is ironically an issue that is often raised as a problem with the waterfall model. Namely that each

phase of the development has to be carried out by designated professionals who only know about the part that they are developing, as well as the parts preceding it. For the CC this ensures that the strength of the PP is not compromised due to knowledge of the desire from the implementation department to enforce their usual development pattern.

One of the problems with the waterfall model is that a locked specification sometimes forces the developers to come up with methods to work around the specification [Sommerville, 2001].

6.3 Reflections

The idea of having a common criteria for evaluating IT products is good in theory, however in practice the Common Criteria cannot hold all security aspects that applies to IT products. To counter this the CC give openings for adding *Extended Components Definitions* (ECD) that follow the exact same structure as the SFR components from CC part 2. But one may ask if it is feasible for PP/ST developers to specify ECDs each time they develop a new PP/ST. With the setup it is possible to reuse previously defined components but the quality of these self-defined components would require some sort of quality stamp, a conformance certificate, to ensure that they are indeed concise and that they provide extra information that does not exist in the SFR components already defined. A way to accommodate this in the implemented program could be to treat the ECDs as components pending certification, but to make it easy for the developers using the developed Toolbox, a module holding all SFR components as well as ECD components as a united components repository has to be developed. That way all ECDs would, to the user of the program, be just as available as the SFRs.

Another issue with the Common Criteria is that it is constantly being developed, this makes it difficult to keep up to date with the newest additions. In earlier versions it was allowed to use SFRs to specify how it is ensured that *SOs on the IT environment* (OE) are dealt with. In the most recent version this is no longer allowed, this makes it difficult for evaluators to evaluate that the OEs are solving the specified problem.

6.4 External Relations

PPs are often written with a different structure of the chapters, but all of them contain the required items, with the output of the implementation as XML it

is possible to create different viewers that can present the PP and ST precisely as wanted. This makes it possible for the PP/ST developer to present the documents in the way they prefer reading it but it also allows the evaluators to present the PP/ST in an easy to evaluate structure that is identical each time a new PP/ST must be evaluated. Furthermore it makes it possible to make different viewers for different parts of the PP/ST. For this thesis a dynamic internet page was created showing the coverage matrix of the coverage of Security Objectives over the Security Problem Definition. The matrix was created by transforming the relevant part of an XML document representing the PP from the Case Study used in this thesis into a PHP array. Furthermore a stylesheet transforming the XML output into SML code to be used for performing well-formedness checks was developed. The output from the transformation can be seen at Appendix A.7 on page 67. All viewers that would be developed for presenting the various parts of the PP/ST must include means of evaluating that the part it presents is indeed valid.

6.5 Summary

This chapter presented the developed prototype with functionality for developing Protection Profiles. Furthermore various extension models of the Toolbox and the relationship between CC and the IT-development life cycle were discussed. Finally reflections on the limitations of the Common Criteria were presented.

Chapter 7

Conclusion

This thesis presents a toolbox providing PP/ST developers with a generalized document that follows a defined structure. The generalized document is presented in a well-known format that suits for sharing information between different systems. Additionally different viewers have been developed presenting key aspects of the document. Finally the theory allowing for transforming output into SML on to which well-formedness functions can be applied, have been showed.

The essence of the Common Criteria is that it has to reach a wide audience. This thesis has held this objective high throughout the development. This goal has been reached through utilizing official document standards and by developing the Toolbox in compliance with best practice of the industry.

One exciting topic that came up late in the development was how to handle the Extended Components Definition. For these components to be usable within any Protection Profile the construction of these must be unified as defined by the CC. A tool that should be added to this toolbox would be such a tool that helps the PP/ST developer to construct Extended Components in conformance to the CC definition.

Appendix A

This Appendix presents different examples of source code as well as the snippets of documents that relate to the development of the program. Furthermore it also presents an example of the output from the program and transformations on this.

A.1 DTD Snippet

This snippet shows the DTD describing the Security Functional Requirements. Section 5.3 on page 37 has been made to describe how the DTD has been interpreted in this thesis.

```
1
2  <!--
3        Functional Paradigm.
4    -->
5
6  <!ELEMENT f-class                (fc-introduction,
7                                    fc-informative-notes,
8                                    f-family+)>
9  <!ATTLIST f-class                name      CDATA     #REQUIRED
10                                  id        ID        #REQUIRED
11                                  patch     IDREF     #IMPLIED>
12
13 <!ELEMENT fc-introduction        (%parasequence;)>
14 <!ELEMENT fc-informative-notes   (%parasequence;)>
15
16 <!-- Functional Family -->
17
18 <!ELEMENT f-family               (ff-behaviour,
19                                   ff-application-notes?,
20                                   ff-user-notes?,
21                                   ff-evaluator-notes?,
22                                   f-component+)>
23 <!ATTLIST f-family               name      CDATA     #REQUIRED
24                                  id        ID        #REQUIRED
25                                  patch     IDREF     #IMPLIED>
26
27 <!ELEMENT ff-behaviour           (%parasequence;)>
28 <!ELEMENT ff-application-notes   (%parasequence;)>
29 <!ELEMENT ff-user-notes          (%parasequence;)>
30 <!ELEMENT ff-evaluator-notes     (%parasequence;)>
31
32 <!-- Functional Component -->
33
34 <!ELEMENT f-component                      (fco-hierarchical?,
35                                             fco-dependencies?,
36                                             fco-rationale?,
37                                             fco-user-notes?,
38                                             fco-evaluator-notes?,
39                                             fco-levelling,
40                                             fco-management*,
41                                             fco-audit*,
42                                             f-element+)>
43 <!ATTLIST f-component                      name      CDATA     #REQUIRED
44                                            id        ID        #REQUIRED
45                                            patch     IDREF     #IMPLIED>
46
47 <!ELEMENT fco-hierarchical                 EMPTY>
48 <!ATTLIST fco-hierarchical                 fcomponent IDREF #REQUIRED
49                                            patch     IDREF     #IMPLIED>
50
51 <!ELEMENT fco-dependencies                 (fco-or|fco-dependsoncomponent)*>
52 <!ATTLIST fco-dependencies                 patch     IDREF     #IMPLIED>
53
54 <!ELEMENT fco-or                           (fco-dependsoncomponent)+>
55 <!ATTLIST fco-or                           patch     IDREF     #IMPLIED>
56
57 <!ELEMENT fco-dependsoncomponent           EMPTY>
58 <!ATTLIST fco-dependsoncomponent           fcomponent IDREF #REQUIRED
59                                            patch     IDREF     #IMPLIED>
60
61 <!ELEMENT fco-rationale                    (%parasequence;)>
62 <!ELEMENT fco-user-notes                   (%parasequence;)>
63 <!ELEMENT fco-evaluator-notes              (%parasequence;)>
64 <!ELEMENT fco-levelling                    (%parasequence;)>
65
66 <!ELEMENT fco-management                   (#PCDATA | xref)*>
67 <!ATTLIST fco-management                   id        ID        #IMPLIED
```

```
68                                              equal     IDREF     #IMPLIED
69                                              patch     IDREF     #IMPLIED>
70
71  <!ELEMENT fco-audit                         (#PCDATA | xref)*>
72  <!ATTLIST fco-audit                         level (minimal|basic|detailed) #REQUIRED
73                                              id        ID        #IMPLIED
74                                              equal     IDREF     #IMPLIED
75                                              patch     IDREF     #IMPLIED>
76  <!--
77        Functional Element
78
79        Text within a functional element differs from the general
80        paragraph in that it incorporates well structured the allowed
81        operations. We also exclude the xref and footnote within these
82        paragraphs. The List entity cannot be used, since it holds
83        general text, so a similar F-ElementList is defined.
84
85    -->
86
87  <!ELEMENT f-element                 (#PCDATA|fe-list|fe-assignment|fe-selection)*>
88  <!ATTLIST f-element                 id        ID        #REQUIRED
89                                      boldfrom  IDREF     #IMPLIED
90                                      patch     IDREF     #IMPLIED>
91
92  <!ELEMENT fe-assignment             (fe-assignmentitem,fe-assignmentnotes?)>
93  <!ATTLIST fe-assignment             id        ID        #IMPLIED
94                                      patch     IDREF     #IMPLIED>
95
96  <!ELEMENT fe-assignmentitem         (#PCDATA|fe-list|fe-assignment|fe-selection)*>
97  <!ATTLIST fe-assignmentitem         patch     IDREF     #IMPLIED>
98
99  <!ELEMENT fe-assignmentnotes        (%parasequence;)>
100
101 <!ELEMENT fe-selection              (fe-selectionitem+,fe-selectionnotes?)>
102 <!ATTLIST fe-selection              id                  ID                  #IMPLIED
103                                     exclusive           (YES|NO)            "NO"
104                                     patch               IDREF               #IMPLIED>
105
106 <!ELEMENT fe-selectionitem          (#PCDATA|fe-list|fe-assignment|fe-selection)*>
107 <!ATTLIST fe-selectionitem          patch     IDREF     #IMPLIED>
108
109 <!ELEMENT fe-selectionnotes         (%parasequence;)>
110
111 <!ELEMENT fe-list                   (fe-item)+>
112 <!ATTLIST fe-list                   patch     IDREF     #IMPLIED>
113
114 <!ELEMENT fe-item                   (#PCDATA|fe-list|fe-assignment|fe-selection)* >
115 <!ATTLIST fe-item                   id        ID        #IMPLIED
116                                     patch     IDREF     #IMPLIED>
```

A.2 IndexFormatter Snippet

```
1  public class IndexFormatter : AbstractFormatter
2  {
3      List<string> stopWords;
4      AbstractLookUp alu;
5
6      Hashtable index;
7      Hashtable classHashtable;
8
9      override public void init(string initString, Hashtable classes) {
10
11         classHashtable = classes;
12
13         stopWords  = new List<string>();
14
15         /*
16             Code for adding stopWords here
17         */
18
19         // Initialize the lookup mechanism used
20         alu = new WordNetLookUp();
21         alu.init(initString);
22
23     }
24
25     override public object format(string position)
26     {
27         index = new Hashtable();
28
29         foreach (object key in classHashtable.Keys)
30         {
31             formatClass(key + "", (fclassinfo)classHashtable[key]);
32         }
33         return index;
34     }
35
36     override internal string formatElement(string key, string unformattedElement)
37     {
38         string elemInfo = "";
39
40            /*
41             * Nothing needed from this level
42             */
43
44         return elemInfo + Environment.NewLine;
45     }
46     override internal string formatComponent(string key, fcomponentinfo fcoi)
47     {
48         string intro = fcoi.fcoLevelling;
49
50         intro = removeSpecialChars(intro);
51
52         foreach (String s1 in intro.Split("_".ToCharArray()))
53         {
54             addOccurence(s1, key);
55         }
56         return "";
57     }
58
59     override internal string formatFamily(string key, ffamilyinfo ffi)
60     {
61         string intro = ffi.ffBehaviour;
62
63         intro = removeSpecialChars(intro);
64
65         foreach (String s1 in intro.Split("_".ToCharArray()))
66         {
67             addOccurence(s1, key);
68         }
69
70         foreach (string fckey in ffi.fcomponent.Keys)
71         {
72             formatComponent(fckey, (fcomponentinfo)ffi.fcomponent[fckey]);
73         }
74
75         return "";
```

```
76      }
77
78
79      override internal string formatClass(string classShortName, fclassinfo fci)
80      {
81          string intro = fci.fcIntroduction;
82
83          intro = removeSpecialChars(intro);
84
85          foreach (String s1 in intro.Split("␣".ToCharArray()))
86          {
87              addOccurence(s1, classShortName);
88          }
89          foreach (string key in fci.cffamily.Keys)
90          {
91              formatFamily(key, (ffamilyinfo)fci.cffamily[key]);
92          }
93          return "";
94      }
95
96      private void addOccurence(string key, string position)
97      {
98          string s = key;
99          s = alu.getShortestForm(s);
100
101         if (!stopWords.Contains(s) && s.Length > 1)
102         {
103             if (index.ContainsKey(s))
104             {
105                 Hashtable ht = ((Hashtable)index[s]);
106                 try
107                 {
108                     ht.Add(position, 1);
109                 }
110                 catch (Exception)
111                 {
112                     int i = (Int32)ht[position]+1;
113                     ht[position] = i;
114                 }
115             }
116             else
117             {
118                 Hashtable ht = new Hashtable();
119                 ht.Add(position, 1);
120
121                 index.Add(s, ht);
122             }
123         }
124     }
125 }
```

A.3 ProfileSchemeContents Snippet

```
1  [Serializable]
2  public struct fcomponentinfoList
3  {
4      public string fcoHierarchical;
5      public string fcoDependencies;
6      public string fcoLevelling;
7      public string fcoManagement;
8      public string fcoAudit;
9      public List<Element> felement;
10 }
11 [Serializable]
12 public struct Element
13 {
14     public string key;
15     public string value;
16 }
17 [Serializable]
18 public struct LinkSPDSO
19 {
20     public string SOname;
21     public string SPDname;
22     public string Rationale;
23 }
24 [Serializable]
25 public struct LinkSOSFR
26 {
27     public string componentName;
28     public string SOname;
29     public string Rationale;
30 }
31 [Serializable]
32 public struct SFRcomponent
33 {
34     public string ComponentShortName;
35     public fcomponentinfoList fComponentInfo;
36 }
37 [Serializable]
38 public struct ConformanceClaims
39 {
40     public string CCconformance;
41     public string PPclaim;
42     public string ConformanceRationale;
43     public string ConformanceStatement;
44 }
45
46 [Serializable]
47 public struct SecurityObjectives
48 {
49     public List<Element> SOdefinition;
50     public List<LinkSPDSO> SORdefinition;
51 }
52 [Serializable]
53 public struct SecurityRequirements
54 {
55     public List<SFRcomponent> SRdefinition;
56     public List<LinkSOSFR> SecurityRequirementsRationale;
57 }
58
59 [XmlRootAttribute("Profile", Namespace="", IsNullable=false)]
60 public class ProfileSchemeContents
61 {
62     public ProfileSchemeContents() { }
63
64     private ConformanceClaims conClaims;
65     private List<Element> spdDef;
66     private SecurityObjectives secuObjec;
67     private SecurityRequirements secReq;
68
69     public ConformanceClaims ConformanceClaims
70     {
71         get { return conClaims; }
72         set { conClaims = value; }
73     }
74
75     public List<Element> SPDdefinition
```

```
76      {
77          get { return spdDef; }
78          set { spdDef = value; }
79      }
80
81      public SecurityObjectives SecurityObjectives
82      {
83          get { return secuObjec; }
84          set { secuObjec = value; }
85      }
86
87      public SecurityRequirements SecurityRequirements
88      {
89          get { return secReq; }
90          set { secReq = value; }
91      }
92  }
```

A.4 ProtectionProfileContents Snippet

```
1
2   [Serializable]
3   public struct PPintroduct{
4       private string PPrefer;
5       private string TOEover;
6
7       public string PPreference
8       {
9           get { return PPrefer; }
10          set { PPrefer = value; }
11      }
12
13      public string TOEoverview
14      {
15          get { return TOEover; }
16          set { TOEover = value; }
17      }
18  }
19  [XmlRootAttribute("ProtectionProfile", Namespace = "", IsNullable = false)]
20  public class ProtectionProfileContents : ProfileSchemeContents
21  {
22      public ProtectionProfileContents() { }
23
24      private PPintroduct ppIntro;
25
26      public PPintroduct PPintroduction
27      {
28          get { return ppIntro; }
29          set { ppIntro = value; }
30      }
31  }
```

A.5 XML output of the PP from the Case Study

```
ï»¿<?xml version="1.0" encoding="utf-8"?>
<ProtectionProfile xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <ConformanceClaims>
    <CCconformance>Common Criteria v. 3.1 Revision 1 Part 2 Conformant
    </CCconformance>
    <PPclaim>Does not conform with any other PP</PPclaim>
    <ConformanceRationale>The PP does not use any extended components
    </ConformanceRationale>
    <ConformanceStatement />
  </ConformanceClaims>
  <SPDdefinition>
    <Element>
      <key>T.UnintendedAccess</key>
      <value>A user may gain unintended access to the TOE</value>
    </Element>
    <Element>
      <key>T.Virus</key>
      <value>A malicious agent may attempt to introduce a virus to the TOE</value>
    </Element>
    <Element>
      <key>A.NoEvil</key>
      <value>It is assumed that the administrators of the TOE do not deliberately
      cause any evil</value>
    </Element>
    <Element>
      <key>A.Physical</key>
      <value>The TOE is assumed to be placed somewhere that requires identification
      to enter</value>
    </Element>
    <Element>
      <key>P.AntivirusDefinitions</key>
      <value>It is dictated by the organization that antivirus definitions must
      be updated on a regular basis</value>
    </Element>
  </SPDdefinition>
  <SecurityObjectives>
    <SOdefinition>
      <Element>
        <key>O.AntivirusUpdate</key>
        <value>The TOE will update Antivirus definitions on a regular basis
        </value>
      </Element>
      <Element>
        <key>O.Virus</key>
        <value>The TOE will detect and take adequate actions against viruses
        </value>
      </Element>
      <Element>
        <key>O.TOEaccess</key>
        <value>The TOE must provide means of how it can be identified who accesses
        the TOE</value>
      </Element>
      <Element>
        <key>OE.Physical</key>
        <value>The IT environment must ensure that nobody can physically access
        the TOE without proper clearance</value>
      </Element>
      <Element>
        <key>OE.NoEvil</key>
        <value>The IT environment where the TOE acts shall ensure that the
        administrators are non-hostile</value>
      </Element>
    </SOdefinition>
    <SORdefinition>
      <LinkSPDSO>
        <SOname>O.AntivirusUpdate</SOname>
        <SPDname>P.AntivirusDefinitions</SPDname>
        <Rationale>The O.AntivirusUpdate ensures that the policy from
        P.AntivirusDefinitions is hold</Rationale>
      </LinkSPDSO>
      <LinkSPDSO>
        <SOname>O.AntivirusUpdate</SOname>
        <SPDname>T.Virus</SPDname>
        <Rationale>The O.AntivirusUpdate ensures that the recent Antivirus
```

```
76          definitions is downloaded and there by making it possible for the TOE to
77          identify new viruses and act accordingly</Rationale>
78        </LinkSPDSO>
79        <LinkSPDSO>
80          <SOname>O.Virus</SOname>
81          <SPDname>T.Virus</SPDname>
82          <Rationale>The O.Virus enables the TOE to take adequate actions whenever
83          a virus is discovered as well as scanning for viruses. </Rationale>
84        </LinkSPDSO>
85        <LinkSPDSO>
86          <SOname>OE.Physical</SOname>
87          <SPDname>T.UnintendedAccess</SPDname>
88          <Rationale>No physical unintended access can be granted to the TOE due to
89          the OE.Physical</Rationale>
90        </LinkSPDSO>
91        <LinkSPDSO>
92          <SOname>OE.Physical</SOname>
93          <SPDname>A.Physical</SPDname>
94          <Rationale>By OE.Physical it is ensured that nobody can access the TOE
95          physically without proper identification</Rationale>
96        </LinkSPDSO>
97        <LinkSPDSO>
98          <SOname>OE.NoEvil</SOname>
99          <SPDname>A.NoEvil</SPDname>
100         <Rationale>The assumption on the administrators being well educated
101         ensures that the assumption A.NoEvil holds</Rationale>
102       </LinkSPDSO>
103       <LinkSPDSO>
104         <SOname>O.TOEaccess</SOname>
105         <SPDname>T.UnintendedAccess</SPDname>
106         <Rationale>Nobody can access the TOE without being identified</Rationale>
107       </LinkSPDSO>
108     </SORdefinition>
109   </SecurityObjectives>
110   <SecurityRequirements>
111     <SRdefinition>
112       <SFRcomponent>
113         <ComponentShortName>fia_uid.2</ComponentShortName>
114         <fComponentInfo>
115           <fcoHierarchical>FIA_UID.1</fcoHierarchical>
116           <fcoLevelling>User identification before any action (FIA_UID.2), requires
117           that users identify themselves before any other action will be allowed
118           by the TSF.</fcoLevelling>
119           <fcoManagement>the management of the user identities.</fcoManagement>
120           <fcoAudit />
121           <felement>
122             <Element>
123               <key>fia_uid.2.1</key>
124               <value>[s "_The_TSF_shall_require_each_user_to_be_successfully
125 _____identified_before_allowing_any_other_TSF-mediated_actions
126 _____on_behalf_of_that_user._"]</value>
127             </Element>
128           </felement>
129         </fComponentInfo>
130       </SFRcomponent>
131     </SRdefinition>
132     <SecurityRequirementsRationale>
133       <LinkSOSFR>
134         <componentName>fia_uid.2</componentName>
135         <SOname>O.TOEaccess</SOname>
136         <Rationale>fia_uid.2 ensures that before a user can use the TOE he
137         must be identified</Rationale>
138       </LinkSOSFR>
139     </SecurityRequirementsRationale>
140   </SecurityRequirements>
141   <PPintroduction>
142     <PPreference>PP illustrating a simple Case Study, April 2, 2007.</PPreference>
143     <TOEoverview>The TOE consist of some parts</TOEoverview>
144   </PPintroduction>
145 </ProtectionProfile>
```

A.6 SPD SO Coverage Matrix online Viewer

Coverage Matrix

Security Objective	Security Problem Definition	Rationale
O.AntivirusUpdate	P.AntivirusDefinitions	The O.AntivirusUpdate ensures that the policy from P.AntivirusDefinitions is hold
O.AntivirusUpdate	T.Virus	The O.AntivirusUpdate ensures that the recent Antivirus definitions is downloaded and there by making it possible for the TOE to identify new viruses and act accordingly
O.Virus	T.Virus	The O.Virus enables the TOE to take adequate actions whenever a virus is discovered as well as scanning for viruses.
OE.Physical	T.UnintendedAccess	No physical unintended access can be granted to the TOE due to the OE.Physical
OE.Physical	A.Physical	By OE.Physical it is ensured that nobody can access the TOE physically without proper identification
OE.NoEvil	A.NoEvil	The assumption on the administrators being well educated ensures that the assumption A.NoEvil holds
O.TOEaccess	T.UnintendedAccess	Nobody can access the TOE without being identified

SOSPD matrix	P.AntivirusDefinitions	T.Virus	T.UnintendedAccess	A.Physical	A.NoEvil
O.AntivirusUpdate	X	X			
O.Virus		X			
OE.Physical			X	X	
OE.NoEvil					X
O.TOEaccess			X		

Figure A.1: Coverage Matrix of SO covering SPD for Case Study

A.7 Transformed SML code

```
1
2  val ppRef = "PP_illustrating_a_simple_Case_Study,
3  _____________April_2,_2007.":PPreference;
4  val toeOv = "The_TOE_consist_of_some_parts":TOEoverview;
5  val ppIntro = (ppRef, toeOv):PPintroduction;
6  val ccConCl = "Common_Criteria_v._3.1_Revision_1_Part_2
7  _____________Conformant_":CCconformanceClaim;
8  val ppClaim = "Does_not_conform_with_any_other_PP":PPclaim;
9  val conRati = "The_PP_does_not_use_any_extended_components
10 _____________":ConformanceRationale;
11 val conStat = "":ConformanceStatement;
12 val conClai = (ccConCl, ppClaim, conRati, conStat
13                ):ConformanceClaims;
14 type SPDdefinition = (SPDname, string) Polyhash.hash_table;
15 val spddef = Polyhash.mkPolyTable(10,error) : SPDdefinition;
16 val sodef = Polyhash.mkPolyTable(10,error) : SOdefinition;
17 val sordef = Polyhash.mkPolyTable(10,error) : SORdefinition;
18 Polyhash.insert spddef (T "T.UnintendedAccess", "A_user_may
19 _____________________gain_unintended_access_to_the_TOE");
20 Polyhash.insert spddef (T "T.Virus", "A_malicious_agent_may
21 _____________________attempt_to_introduce_a_virus_to_the_TOE");
22 Polyhash.insert spddef (A "A.NoEvil", "It_is_assumed_that
23 _____________________the_administrators_of_the_TOE_do_not
24 _____________________deliberately_cause_any_evil");
25 Polyhash.insert spddef (A "A.Physical", "The_TOE_is_assumed
26 _____________________to_be_placed_somewhere_that_requires
27 _____________________identification_to_enter");
28 Polyhash.insert spddef (P "P.AntivirusDefinitions", "It_is
29 _____________________dictated_by_the_organization_that
30 _____________________antivirus_definitions_must_be_updated
31 _____________________on_a_regular_basis");
32 Polyhash.insert sodef (O "O.AntivirusUpdate", "The_TOE_will
33 _____________________update_Antivirus_definitions_on_a
34 _____________________regular_basis_");
35 Polyhash.insert sodef (O "O.Virus", "The_TOE_will_detect_and
36 _____________________take_adequate_actions_against_viruses_");
37 Polyhash.insert sodef (O "O.TOEaccess", "The_TOE_must_provide
38 _____________________means_of_how_it_can_be_identified_who
39 _____________________accesses_the_TOE");
40 Polyhash.insert sodef (OE "OE.Physical", "The_IT_environment
41 _____________________must_ensure_that_nobody_can_physically
42 _____________________access_the_TOE_without_proper_clearance");
43 Polyhash.insert sodef (OE "OE.NoEvil", "The_IT_environment
44 _____________________where_the_TOE_acts_shall_ensure_that
45 _____________________the_administrators_are_non-hostile");
46 Polyhash.insert sordef (( P "P.AntivirusDefinitions" ,
47                           O "O.AntivirusUpdate" ) ,
48                           "The_O.AntivirusUpdate_ensures_that_the
49 _____________________policy_from_P.AntivirusDefinitions_is_hold");
50 Polyhash.insert sordef (( T "T.Virus" , O "O.AntivirusUpdate" )
51                         , "The_O.AntivirusUpdate_ensures_that_the
52 _____________________recent_Antivirus_definitions_is_downloaded
53 _____________________and_there_by_making_it_possible_for_the_TOE
54 _____________________to_identify_new_viruses_and_act_accordingly");
55 Polyhash.insert sordef (( T "T.Virus" , O "O.Virus" ) , "The_O.Virus
56 _____________________enables_the_TOE_to_take_adequate_actions
57 _____________________whenever_a_virus_is_discovered_as_well
58 _____________________as_scanning_for_viruses._");
59 Polyhash.insert sordef (( T "T.UnintendedAccess" , OE "OE.Physical" )
60                         , "No_physical_unintended_access_can_be
61 _____________________granted_to_the_TOE_due_to_the_OE.Physical");
62 Polyhash.insert sordef (( A "A.Physical" , OE "OE.Physical" ) ,
63                           "By_OE.Physical_it_is_ensured_that_nobody
64 _____________________can_access_the_TOE_physically_without_proper
65 _____________________identification");
66 Polyhash.insert sordef (( A "A.NoEvil" , OE "OE.NoEvil" ) , "The
67 _____________________assumption_on_the_administrators_being_well
68 _____________________educated_ensures_that_the_assumption
69 _____________________A.NoEvil_holds");
70 Polyhash.insert sordef (( T "T.UnintendedAccess" , O "O.TOEaccess" )
71                         , "Nobody_can_access_the_TOE_without_being
72 _____________________identified");
73 val secObj = (sodef, sordef);
74 val extComp="";
75 val fcompMap = Polyhash.mkPolyTable(10,error) : fcomponentMap;
76 val felementMap = Polyhash.mkPolyTable(10,error) : felement;
```

```
77  Polyhash.insert felementMap (Element "fia_uid.2.1", [s "_The_TSF
78  ____________________________shall_require_each_user_to_be_successfully
79  ____________________________identified_before_allowing_any_other_TSF-mediated
80  ____________________________actions_on_behalf_of_that_user._"]);
81  Polyhash.insert fcompMap ( Component "fia_uid.2",( "FIA_UID.1", "",
82                              "User_identification_before_any_action
83  ____________________________(FIA_UID.2),_requires_that_users_identify
84  ____________________________themselves_before_any_other_action_will_be
85  ____________________________allowed_by_the_TSF.", "the_management_of_the
86  ____________________________user_identities.", "", felementMap) );
87  val secReqRat = Polyhash.mkPolyTable(10,error)
88                              : SecurityRequirementsRationale;
89  Polyhash.insert secReqRat ((Component "fia_uid.2", O "O.TOEaccess" ),
90  "fia_uid.2_ensures_that_before_a_user_can_use_the_TOE_he_must_be
91  ____________________________identified");
92  val pp = (ppIntro, conClai, spddef, secObj, extComp, secReqRat)
93                              :ProtectionProfile;
```

A.8 SML functions

```
(* An assumption may never be covered by a TOE Objective *)
fun assumptionTest [] = true
|   assumptionTest ((A _,O _)::_) = false
|   assumptionTest (_::xs) = test xs
;

fun eKeyHelper [] = []
|   eKeyHelper ((x1,x2)::xs) = x1::eKeyHelper xs;

fun eKeyHelperRev [] = []
|   eKeyHelperRev ((x1,x2)::xs) = x2::eKeyHelperRev xs;

infix member
fun x member [] = false
|   x member (y::ys) = x=y orelse x member ys;

infix cover
fun xs cover ys = Set.equal (Set.fromList xs, Set.fromList ys);

fun extractKeys def =
        let val xs = Polyhash.listItems def in
            eKeyHelper xs
        end;

fun extractFirstKeyComp def =
        let val xs = Polyhash.listItems def in
            eKeyHelper (eKeyHelper xs)
        end;

fun extractSecondKeyComp def =
        let val xs = Polyhash.listItems def in
            eKeyHelperRev (eKeyHelper xs)
        end;

fun isInDom def n = n member extractKeys def;

fun is_wellFormedSPD (spd:SPDdefinition) = true;

fun is_wellFormedSO spd ((sod, sord):SecurityObjectives) =
            extractFirstKeyComp sord cover extractKeys spd andalso
            extractSecondKeyComp sord cover extractKeys sod andalso
            assumptionTest (extractKeys sord)

;
fun is_wellFormedSR ((srd, srr):SecurityRequirements) =
                    extractKeys srd cover extractFirstKeyComp srr
;
fun is_wellFormedPP ((_,spd,so,_,sr):ProtectionProfile) =
                    is_wellFormedSPD spd andalso
                    is_wellFormedSO spd (so) andalso
                    is_wellFormedSR sr
;

fun is_wellFormedPP ((_,_,spd,so,_,sr):ProtectionProfile) =
                    is_wellFormedSPD spd andalso
                    is_wellFormedSO spd (so)
;
```

Appendix B

Source Code - CD

The Source Code has been provided on the attached CD-rom.

Bibliography

[Cockburn, 2007] Cockburn, A. (2007). Alistair cockburn, http://alistair.cockburn.us/.

[Criteria, 2006] Criteria, C. (2006). Common criteria for information technology security evaluation. CCMB 2006-09-001 Version 3.1 Revision 1.

[Hansen and Rischel, 1999] Hansen, M. R. and Rischel, H. (1999). *Introduction to Programming using SML*. Addison-Wesley, first edition.

[Microsoft, 2007] Microsoft (2007). Microsoft .net framework 2, http://msdn2.microsoft.com/en-gb/netframework/default.aspx.

[Pfleeger and Pfleeger, 2003] Pfleeger, C. P. and Pfleeger, S. L. (2003). *Security in Computing*. Prentice Hall, third edition.

[Portal, 2007] Portal, C. C. (2007). Common criteria http://www.commoncriteriaportal.org/.

[REGEX, 2007] REGEX (2007). Regular-expressions.info, http://www.regular-expressions.info/.

[Sommerville, 2001] Sommerville, I. (2001). *Software Engineering*. Addison-Wesley, sixth edition.

[SPARTA, 2007] SPARTA (2007). Common criteria toolbox http://cctoolbox.sparta.com/.

[WordNet, 2007] WordNet (2007). Wordnet, http://wordnet.princeton.edu/.