Examensarbete

Design and Implementation of an Environment to Support Development of Methods for Security Assessment

Master’s thesis in Information Theory by

Johan Bengtsson and

Peter Brinck

LiTH-ISY-EX--08/4022--SE

Linköping, 2008
Design and Implementation of an Environment to Support Development of Methods for Security Assessment

Master's thesis in Information Theory, at Linköping Institute of Technology

by

Johan Bengtsson

and

Peter Brinck

LiTH-ISY-EX--08/4022--SE

Supervisor: Jonas Hallberg, FOI
Examiner: Viiveke Fåk
Linköping, 2008-Mar-07
**Abstract**

There is no debate over the importance of IT security. Equally important is the research on security assessment; methods for evaluating the security of IT systems. The Swedish Defense Research Agency has for the last couple of years been conducting research on the area of security assessment. To verify the correctness of these methods, tools are implemented.

This thesis presents the design and implementation of an environment to support and aid future implementations and evaluations of security assessment methods. The aim of this environment, known as the New Tool Environment, NTE, is to assist the developer by facilitating the more time consuming parts of the implementation. A large part of this thesis is devoted to the development of a database solution, which results in an object/relational data access layer.

**Keywords:** IT Security, Security Assessment, Security Assessment Tool
Abstract

There is no debate over the importance of IT security. Equally important is the research on security assessment; methods for evaluating the security of IT systems. The Swedish Defense Research Agency has for the last couple of years been conducting research on the area of security assessment. To verify the correctness of these methods, tools are implemented.

This thesis presents the design and implementation of an environment to support and aid future implementations and evaluations of security assessment methods. The aim of this environment, known as the New Tool Environment, NTE, is to assist the developer by facilitating the more time consuming parts of the implementation. A large part of this thesis is devoted to the development of a database solution, which results in an object/relational data access layer.
## Contents

1  **Introduction** ........................................................................................................ 1  
   1.1  Motivation ........................................................................................................ 1  
   1.2  Problem Formulation ..................................................................................... 2  
   1.3  Contributions .................................................................................................. 3  
   1.4  Disposition ..................................................................................................... 3  
   1.5  Terminology .................................................................................................. 3  

2  **Background** ...................................................................................................... 5  
   2.1  Security Assessment ...................................................................................... 5  
   2.2  Development History .................................................................................... 5  
   2.3  Related Work .................................................................................................. 7  
   2.4  XMASS ........................................................................................................... 8  

3  **Design of the New Tool Environment** ............................................................. 11  
   3.1  Requirement Analysis .................................................................................... 11  
   3.2  Structure Overview ...................................................................................... 17  

4  **Implementation of NTE** .................................................................................. 21  
   4.1  Plugin Handling ............................................................................................. 21  
   4.2  Tool Plugins .................................................................................................. 22  
   4.3  Database Plugins ........................................................................................... 22  
   4.4  Common Library ............................................................................................ 24  
   4.5  Project Handling ............................................................................................ 24  
   4.6  Aggregation & Presentation ........................................................................... 30  
   4.7  Handling of Predefined Input ....................................................................... 30  

5  **Implementation of the Database Plugin** .......................................................... 33  
   5.1  Plugin Structure ............................................................................................ 33  
   5.2  Table Structure ............................................................................................ 36  
   5.3  Operations on the Database ......................................................................... 38  

6  **Implementation of XMASS** ............................................................................. 41  
   6.1  Design Focus .................................................................................................. 41  
   6.2  Graphical User Interface .............................................................................. 41  
   6.3  Method Improvements .................................................................................. 42  
   6.4  Result Aggregation and Presentation .......................................................... 42  

7  **Conclusions** .................................................................................................... 45  
   7.1  Discussion ...................................................................................................... 45  
   7.2  Results .......................................................................................................... 47  
   7.3  Suggestions for Future Improvements ......................................................... 47  

**Appendix A - User Guide for NTE & XMASS** ..................................................... 51  
**Appendix B - Common Library** .......................................................................... 69
| Figure 1. | The structure of XMASS. (Hallberg et al, 2006). | 9 |
| Figure 2. | Use cases from a developer perspective. | 12 |
| Figure 3. | Use cases from a user perspective. | 14 |
| Figure 4. | A generalized view of an assessment tool. | 15 |
| Figure 5. | Expansion of the Manage Systems use case. | 15 |
| Figure 6. | Comparison of two implemented methods. | 16 |
| Figure 7. | Expansion of the Manage Predefined Input use case. | 17 |
| Figure 8. | Schematic view of the structure of the environment. | 17 |
| Figure 9. | The structure of requirement collections. | 19 |
| Figure 10. | Class diagram of the tool plugin interface. | 22 |
| Figure 11. | Class diagram of the database interfaces. | 23 |
| Figure 12. | Class diagram of the NTE file classes. | 25 |
| Figure 13. | Sequence diagram of the create system operation. | 26 |
| Figure 14. | Sequence diagram of the open system operation. | 27 |
| Figure 15. | Sequence diagram of the save system operation. | 28 |
| Figure 16. | Sequence diagram of the delete system operation. | 28 |
| Figure 17. | Sequence diagram of the export operation. | 29 |
| Figure 18. | Class diagram of requirement collections. | 31 |
| Figure 19. | Class diagram of filter functional requirements. | 31 |
| Figure 20. | Schematic view of the structure of the database plugin. | 34 |
| Figure 21. | A simple class structure and its corresponding table structure. | 36 |
| Figure 22. | An example of a relation between two classes. | 37 |
| Figure 23. | An example of a class containing a list. | 37 |
| Figure 24. | An example of a class containing a dictionary. | 37 |
| Figure 25. | Result aggregation and presentation in XMASS. | 43 |
Introduction

During the last decade, the number of computers and computer users has drastically increased. Along with a wide-spread use of the Internet comes an increase in the number of security threats that needs to be addressed. Besides being a help when it comes to spreading news and knowledge, the Internet is also used by evil-minded people for spreading viruses. Having a massive amount of computers interconnected also increases the risk for intrusion attempts. The increase in the number of security threats leads to a greater need for ways to assess the level of security in various systems.

1.1 Motivation

The Swedish Defense Research Agency, FOI, has for the last couple of years been developing several methods for the assessment of IT security. Most important of these are MASS, Method for Assessment of System Security, (Andersson, 2005) and its successor XMASS, eXtended Method for Assessment of System Security, (Hallberg et al, 2006).

Due to the complexity of assessment methods, software implementations are often necessary in order to be able to evaluate the methods. Methods without software implementations tend to stay in the state of being just suggestions on paper. Different software implementations have been developed to illustrate and evaluate the developed methods. During the last years the implementation of MASS, called ROME2, has been updated as the method has been developed. Today there are several different branches of the software supporting different extensions of MASS and thereby lacking compatibility with each other. The software development has dropped behind the method development and there is a need to update ROME2 to support the successor XMASS.
1.2 Problem Formulation

The original problem formulation for this thesis, as given by FOI, was to

- develop a database prototype for managing data of relevance to XMASS,
- modify ROME2 to use the database and
- extend ROME2 to include the functionality of XMASS.

However, after a careful study of the problem and after discussions with FOI the problem formulation was revised as additional needs were discovered.

There is no general environment to use when implementing tools for security assessment methods. The implementation is today equivalent with implementing completely new software from scratch. A need has been found for a development environment that can ease the implementation of assessment tools by reducing the work needed to implement the basic functionality.

With such a developing environment the need for a tool implementing XMASS still remains, but it can then be regarded as an example of a tool implementation whose purpose is to evaluate the development environment. This tool implementation should serve a guide for future implementations.

Today, there is no functionality in ROME2 to aggregate over the assessment result, that is, to apply a function over a set of results. An aggregation is useful, not only when it comes to evaluating the assessment results of a system, but also when it comes to evaluating the performance of the method itself. Manual aggregation is time consuming and the process also involves third-party software to evaluate the results. Thus, aggregation functionality would save time when assessing systems and it would also help evaluating the security assessment method itself. Along with the need for aggregation functionality comes the need for improved presentation of the assessment results. Aggregation functionality alone without good possibilities to present the results would not eliminate the need of third-party software.

To sum up the discussion, the aim of the work presented in this thesis is to design and implement:

- an environment supporting the future implementations of tools for security assessment,
- a suitable database solution,
- support for the aggregation over assessment results,
- a presentation view of the aggregated result and
- a security assessment tool implementing XMASS in this new environment.
1.3 Contributions
The work described in this thesis contributes to the research on security assessment methods by reducing the time spent on implementing tools, with the ambition that more methods get implemented and evaluated. It also provides means for improved evaluation and comparisons of assessment methods.

1.4 Disposition
Chapter 2 explains the concept of IT security and security assessment. This chapter also presents the previous relevant projects and a description of XMASS.

Chapter 3 describes the design process.

Chapter 4 explains the structure of the developed application.

Chapter 5 describes the implementation of a database plugin and chapter 6 the implementation of a tool plugin.

Chapter 7 presents the results of the thesis and a discussion around these results.

1.5 Terminology

**Dynamic-link library (DLL)**
A DLL-file is a shared library in the Microsoft Windows operating system.

**Environment**
In this thesis, the word environment is used to describe the developer’s view of the application. This view involves, apart from the application itself, the interfaces and auxiliary components such as the database and shared libraries.

**Security assessment method (method)**
A theoretical description of a process for assessing the security of a system.

**Tool**
A realization of a security assessment method.

**Security**
While talking about assessing the security of a system, the term security is in this thesis used in the meaning of IT security.

**System**
When used in the context of “assessment of system security” it is defined as a network of related entities. In the design and implementation of NTE the term is defined as a configuration of resources.
2 Background

In this chapter the relevant background for this thesis is provided. A brief introduction to the area of security assessment is followed by a walkthrough of the previous research leading up to this thesis. The last section gives an overview of XMASS and explains the various parts of the method.

2.1 Security Assessment

The meaning of security differs depending on the context it appears in; hence it is hard to give an explicit definition which is suitable for all contexts. A range of different definitions for security are used today. One definition provided by the Swedish Standards Institute (2004) defines security as the ability to maintain desired confidentiality, integrity and availability.

The purpose of a security assessment is to give an indication of how well a system meets specified criteria. Although perfect security is the ultimate goal for a system, an assessment cannot guarantee any level of security, but it can provide a basis for confidence in the assessed system (Bishop, 2003).

To be able to get an estimate of how well a security assessment method performs and to identify necessary method improvements, it is important to evaluate the method. Most security assessment methods consist of calculations that tend to get quite complex as assessed systems grow in size. It is for example a quite laborious task to evaluate such a complex security assessment method as XMASS by hand. Hence the calculation power of a computer is needed to assess non-trivial systems.

2.2 Development History

During the past years both assessment methods and supporting tools have been developed at the Swedish Defense Research Agency, FOI. The methods and tools most relevant for this thesis are described in the following sections.
2.2.1 Methods of Assessment
The research at FOI has resulted in the following methods of security assessment.

**The Heimdal Framework**
The Heimdal Framework was introduced by Bond and Påhlsson (2004). The framework includes two main tasks. The first task is to model system components by using predefined entities, while the other task is to use the model to assess the security of the system components. The framework was built to be as general as possible, so a collection of parameters have to be set in order to transform the framework into a method.

**CAESAR**
The CAESAR method was first introduced by Peterson (2004) and further developed by Hallberg et al (2004). It was made to illustrate the possibilities of structural system security assessment methods. The method estimates a security level of the entire system based on the security level of the included components and the component relations within the system. CAESAR is presented as an illustrative method that is far from complete.

**MASS**
The Method for Assessment of System Security, MASS, was introduced by Andersson (2005) and further developed by Andersson & Hallberg (2006).

The method is based on security-relevant characteristics of components which are modeled by a set of security features. Entity relations are modeled by special functions to capture the relations between the security features of the components. The security of individual entities is assessed based on the components and the relations. The results of the individual assessments are then used to aggregate measures of the overall security of the system.

**XMASS**
The eXtended Method for Assessment of System Security, XMASS, was introduced by Hallberg et al (2006). XMASS is based on the Method for Assessment of System Security, MASS.

The main improvement is the integration of the input preparation that was previously needed in MASS. The calculations of security profiles and filter profiles have become an integrated part of the method. Also the modeling of inter-entity relations has been integrated in the method. These new integrated parts are based on methods for the preparation of input to MASS (Andersson & Hallberg, 2006), but have been both improved and extended to fit as an integrated part of the method. The method is further described in section 2.4.

2.2.2 Supporting Tools
Supporting tools have been implemented to evaluate some of the assessment methods that have been developed at FOI.
Heimdal Security Evaluator 3000 .NET
A software implementation based on the Heimdal Framework called *Heimdal Security Evaluator 3000 .NET* was presented by Bond & Pahlsson (2004). The program is an automation tool used for generating assessments.

ROME
The ROME software is an implementation of the CAESAR method for assessment of system security described in section 2.2.1. The software was implemented to illustrate and evaluate the CAESAR method (Hallberg et al, 2004). In ROME it is possible to create and save system models as well as modifying and viewing them. The model evaluation is done in real-time, which means that alternations of components and relations immediately affect other components and the overall security level.

ROME2
ROME2 is an improvement of the first version of ROME which includes some parts of the Heimdal Security Evaluator 3000 .NET. This version of ROME is based on MASS instead of the CAESAR method which the original version was based on. Additional software like for example Microsoft Excel in combination with the Relation Converter for Rome (Andersson & Hallberg, 2006), is used to create the input needed to assess the security of a system. The implementation has, since the first implementation of ROME2, been further developed to reflect updates of the MASS method and to add new features. The result of this development is a range of different versions of the program with varying compatibility.

2.3 Related Work
Other methods of security assessment have been developed by other parties over the years. Hallberg et al (2006) classifies five different security assessment methods, not developed by FOI, while evaluating their framework for system security assessment called Crossroads.

System Vulnerability Index (SVI)
The SVI method was introduced by Alves-Foss & Barbosa (1995). The method considers general system characteristics with the goal to find characteristics that are general enough to provide system independent values. A specific SVI would then, based on a specific set of SVI rules, reveal the vulnerability. The method is not based on structural modeling of systems. It relies on the validity of the specified SVI rules and their equal importance.

Real-time Risk Assessment with Network Sensors and Hidden Markov Models
The method, presented by Årnes et al (2006), distinguishes itself from the other methods described, by assessing system security in real-time. It does not produce security assessment as an end result but is rather used for risk management and network monitoring. The security assessment is performed using hidden Markov models to, for example, model workstations and servers.
Qualitative and Quantitative Analytical Techniques for Network Security Assessment

The framework, proposed by Clark et al (2004), is based on a multi-stage attack modeling framework, supporting the modeling of vulnerabilities, network structures and attacker capabilities.

Security Measurement (SM) Framework

The Security Measurement Framework, presented by Wang and Wulf (1997), aims to estimate scalar security values which correspond to high-level security attributes. A method specific modeling has been developed based on the use of physical entities, logical entities, security functions and security qualities. A decomposition method is used to retrieve measurable attributes from abstract concepts of security. The decomposition results in a tree with measurable security attributes as leaves.

Analyzing the Security and Survivability of Real-time Control Systems

The method, proposed by Oman et al (2004), is targeted towards systems for supervisory control and data acquisition (SCADA), but might also be generally applicable. Threats based on intentional attacks against SCADA systems are targeted rather than the security values of systems. Network models are transformed into graph representations, which enable the use of well-known and well-defined graph algorithms.

2.4 XMASS

The modeling language of XMASS includes entities and relations. There are two types of entities – traffic generators and traffic mediators. A traffic generator can represent a server, a workstation or something else producing traffic whereas a traffic mediator forwards traffic between generators and can represent for example a hub, router or firewall. Two types of relations are defined in XMASS – physical relations and logical relations. The physical relations describe bidirectional associations between entities through physical means, such as wired or wireless communication, while the logical relations describe unidirectional dependencies and communication between entities.

The structure of XMASS is illustrated in figure 1. To describe the security relevant qualities of systems, XMASS uses a set of security features with corresponding numeric values in the range of [0, 1], describing the level of fulfillment for each security feature. These security values are referred to as security profiles when describing the security characteristics of entities, and filter profiles when describing the filtering capabilities of traffic mediators.

The security profiles express the independent security characteristics of an entity. To account for the effects of inter-entity relations a system-dependent security profile is calculated for each entity. The system-dependent security profiles are used for calculating the system-level security values, which describe the overall security of the system.
Calculation of Profiles

The security values of the profiles are in the range \([0, 1]\), where 0 means that the security quality, described by the security feature, is not addressed by the entity and 1 means that the security quality is perfectly handled.

Profiles are created from *profile templates* which are created by prioritizing a set of requirements for each security feature. The prioritization is performed by deciding the relative importance among each pair of requirements, using the method for criteria weighting from the Analytic Hierarchy Process, AHP (Saaty, 1994). Profiles are then created from the templates by stating how well each requirement is met by the considered entity. For a security profile these values are referred to as the *fulfillment values*, whereas for a filter profile they are known as the *filter capability values*.
3 Design of the New Tool Environment

This chapter describes the design of the New Tool Environment, NTE. The first section identifies the requirements on NTE, while the second section describes the structure of NTE.

3.1 Requirement Analysis

As can be seen in the previous chapter, the development in the area of security assessment is an ongoing process. The methods are constantly refined requiring the implementations to be updated, rewritten or developed from scratch. Moreover, the research on the methods is often branched into different approaches, which leads to branching in the program development resulting in a multitude of different versions. This results in difficulties to evaluate the combined effects of the research, since it can be hard to keep track of the different versions.

The development of assessment methods is still in an early state which leads to the conclusion that the main group of users of tool implementations is the researchers who want to evaluate and compare different assessment methods and their implementations as tools. In many cases the user of the assessment tool and the programmer, implementing the tool, are the same person. Even so, NTE is designed from the perspective of both the user and the developer in order to identify all needs and requirements. To emphasize and distinguish the perspectives of the user and the developer, NTE is referred to as either an application or an environment, where the latter includes the developer specific parts. The aim of the environment is to assist the developer implementing tools while the aim of the application is to ease the work of the user evaluating assessment methods and tools. From this reasoning the two first requirements on NTE are formulated as:

**Req 1:** NTE as an environment should assist the developer implementing tools for security assessment.
CHAPTER 3 - DESIGN OF THE NEW TOOL ENVIRONMENT

Req 2: NTE as an application should aid the user in the evaluation of methods for the assessment of system security and the tools implementing these methods.

To bridge the gap between the two perspectives of NTE, the tools implemented with help from the environment are defined to be plugins to the application. This is captured by the following requirement:

Req 3: The application should provide functionality to run security assessment tools implemented as plugins.

3.1.1 Developer Perspective
Implementing a method can be a tedious process. Focus is divided between the implementation of the actual method and basic tasks like file handling and user interface solutions. This, in combination with limited time to spend on the development, may lead to poor code quality and low extensibility. Programming always requires a certain amount of work, but if the work can be assisted as much as possible the developer can focus on implementing the assessment method. The use cases identified from a developer perspective are displayed in figure 2. The main task of the developer is to implement a method for security assessment. To accomplish this task the developer is aided by utilizing the functionality of NTE.

Figure 2. Use cases from a developer perspective.

NTE is to provide the basic tools needed and simple interfaces to communicate with these tools. Having well defined interfaces to work against can also assist in the design of a program. The following statement describes the importance with well defined interfaces:

“The designer of a module should provide a client with all the information necessary to use the module effectively and nothing more, and should provide the implementer with the information needed to code that module and nothing more.” (Drake, 1998)

Although this statement describes the views of an interface from the perspective of two different persons it can also be seen as describing two interfaces, where the client and the implementer are the same person, i.e. the developer of the assessment method. The first part can be regarded as describing the interface of auxiliary modules, such as a database, where the developer only needs to know how to use the module. This interface should be well defined in order to let the developer use the module effortlessly without worrying about the underlying implementation. The
second part describes the interface to the tool itself, which also needs to be well defined in order to let the programmer spend less time on figuring out the control flow of the program and instead concentrate on writing functions for certain specific events, defined in the interface. The interface becomes a form of guideline, assisting the programmer in the implementation. The reasoning above is summarized in the following requirements:

**Req 4:** The environment should provide a well defined interface to guide the implementation of tool plugins.

**Req 5:** The environment should provide a well defined interface to the database.

A programmer normally spends a great deal of time designing and implementing functionality to store data on disk. The functionality includes such time consuming tasks as converting data from the object based representation of the program to a suitable representation on disk. One way to reduce the required effort is to implement a data access layer between the database and the environment to assist when storing and fetching objects. The programmer should be able to send objects directly to the layer, which handles the storing in the database. In the same way, the layer should be able to rebuild the stored object when requested. The data access layer should also be transparent to the developer and not impose on the class definitions with artefacts such as database IDs.

As suggested by Drake (1998), the main advantage with a clear distinction between the interface and implementation is the possibility to write, test and debug the modules, which make up the application, independently. Separating the database implementation from NTE should benefit from these advantages and also allow for future improvements and optimizations on the database. The requirements on the data access layer are thus:

**Req 6:** The data access layer should operate on objects and contain functionality for storing, fetching and deleting objects from the database.

**Req 7:** The data access layer should be transparent to the developer.

**Req 8:** The database implementation should be separated from NTE.

The design of the user interface is an important part of the implementation process of a method, since a well designed user interface can make the evaluation process more efficient. NTE must not limit the possibilities of the tool implementations but rather assist the developer in the design. One way of assisting the developer is to provide a library of components which are to be shared by all tool implementations. The aim of the library is to streamline the design process while at the same time ensuring some form of conformity between the implementations. This is captured by the following requirement:

**Req 9:** NTE should provide the developer with a library of components, useful for the implementation of security assessment methods.
3.1.2 User Perspective

As stated above, from the user’s point of view NTE has the role of being an application for evaluating methods and tools for security assessment. The role of the user can be viewed from two perspectives as:

- to evaluate methods by observing changes in the result upon changes in the input and
- to compare methods by comparing the results from the methods on the same set of input.

These tasks are identified as the use cases *Evaluate Method* and *Compare Methods*, respectively, and can be seen in figure 3. The figure also contains the generalized use cases *Manage Systems* and *Utilize Predefined Input*, which are explained further down in this section.

![Figure 3. Use cases from a user perspective.](image)

**Evaluation of Methods**

In figure 4 a generalized view of an assessment tool is depicted, illustrating the steps from input to output. The tool is assumed to perform some form of *system modeling*, where the *input* is transformed into *systems* and *resources*. Resources are a collection of components derived from the input, like for example the security characteristics of a specific computer or firewall. A system is a configuration of these resources in a way that is meaningful for the assessment method, such as a model of how the computers are connected in a network. To further improve the usability of the application and the possibilities to experiment with different configurations, the systems are grouped into *projects*. A project is hence defined as a collection of related systems sharing common resources. The tool is also assumed to have a *calculation* step, where the system and the relevant resources are transformed into an *assessment result*, which is the output of the tool.
The notion of systems is introduced to facilitate the evaluation of methods. The idea is to let the user store different configurations of the resources in order to compare them and draw conclusions from how they affect the results. The application needs to give the user the possibility to perform basic operations on these systems, such as creating, opening, saving and deleting. These needs are expressed by the expansion of the Manage Systems use case in figure 5, which also includes the export operation explained further down in this section. The needs result in the following requirements:

- **Req 10:** NTE should allow for tools to operate on systems.
- **Req 11:** Systems and resources should be grouped into projects.
- **Req 12:** Functionality to create, open, save and delete systems should be provided by NTE.

The grouping of systems into projects not only gives the possibility for systems to have common resources, but also gives the user a possibility to get a more organized view of the systems. One project can for example model a specific attribute of the assessment method, with the systems within showing different aspects of that attribute.

There might be occasions when there is a need to copy some, or all, of the resources in one or more projects to another. Such an occasion could for example occur if the user wants to create a new project containing a subset of the shared data from earlier projects. All the relevant data should be exported from the source projects to the target project; hence there should not be any dependencies between the different projects. These needs are summarized in requirement 13.
Req 13: The application should provide functionality to export some or all systems and/or resources from one project to a new or existing project.

To further aid the user in the evaluation of methods, the application should be able to aggregate over a set of results from an assessment method and present the aggregated result in a way from which the user can draw conclusions about how the security is affected.

Req 14: The application should provide the user with functionality to aggregate over the results from a method and present this in a suitable way.

Comparison of Methods

As stated in the use cases, one of the user’s tasks is to compare assessment methods. Therefore NTE needs to be flexible enough to manage to work with different methods. The feature of having assessment tools implemented as plugins gives the user the possibility to create projects using different assessment methods. This solution gathers all assessment tools into one application, which facilitates for the user who does not have to use a combination of different applications to reach an assessment result.

When identifying the requirements for the use case of comparing methods, the view depicted in figure 6 is used. In this view, two tool implementations perform a security assessment based on the same predefined input and produce results which are compared to each other.

![Figure 6. Comparison of two implemented methods.](image)

If all tools share the same input, the result should be easier to compare. This input should constitute the base of the system modeling in each assessment tool, which leads to the following requirement:

Req 15: The application should support the functionality of applying predefined data as input to the tools.

In figure 7 the generalized use case Utilize Predefined Input is expanded, identifying not only the need for applying, but also for switching and managing input. Switching input is important, because it allows the user to try different sets of input on the same system. The Manage use case states that NTE should let the user create and alter the input from within the application. These needs are covered by the following requirements:

Req 16: The user should be able to manage predefined input.

Req 17: The user should be able to switch to another set of input for a system.
3.2 Structure Overview

To summarize the requirement analysis, the structure of NTE is shown in figure 8 and described in the following sections.

**The Application**

The part of NTE which is visible for the user is in the figure referred to as the application. The main tasks of the application are to provide functionality for managing projects and predefined input and to provide a workspace to be used by the assessment tools.

**Plugins**

There are two types of plugins presented in the figure – tool plugins and database plugins. A tool plugin is an implementation of a security assessment method, while a database plugin contains the functionality needed to handle a database. NTE should be able to handle a multitude of these two types of plugins and thereby support future additions of plugins of the mentioned types. Plugins should be selected for each project, which results in a structure where all systems within a project are using the same set of plugins.

**The Interfaces**

Three different interfaces are needed to enable communication between the application, the tool plugin and the database plugin. The tool interface is to be used
by the application for communication with the tool plugin. As can be seen in Figure 8, two different database interfaces are suggested. Both the application and the tool plugin need to communicate with the database, but they have different needs when it comes to functionality. To make the database interface as simple and straightforward as possible to use for the tool plugin developer, the application and the tool plugin use different interfaces which should include the needed functionality and nothing more.

The Common Library
The common library contains various components intended for use by the tool plugins. The library should contain GUI, Graphical User Interface, related components which can be used to reduce the amount of time the tool plugin developer needs to spend on GUI implementation.

Project File
A project file is suggested to encapsulate the project related data which needs to be stored. To make a clear distinction between different projects, each project should have its own project file and thereby also its own database file.

NTE needs information about the tool and database plugin used by each project in order to open systems from a project. It also needs information about the systems within a project in order to list available systems without being forced to activate the used plugins. The conclusion is that each project file should be an encapsulation of the following:

- Information regarding the used tool and database plugin
- Information about the systems within the project
- A database file which contains the project and its systems

A clear distinction between different projects together with the functionality to export systems to new projects should give the possibility to for example send one of the systems within a project to another user without being forced to send all.

Predefined input
The structure of the predefined input is based on *Krav på SäkerhetsFunktioner*, KSF (Swedish Armed Forces, 2004). In KSF, the properties and qualities of an IT system are described by a set of *security features*. Examples of such features are found in table 1.

<table>
<thead>
<tr>
<th>Security feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access control</td>
</tr>
<tr>
<td>Security logging</td>
</tr>
<tr>
<td>Intrusion detection</td>
</tr>
<tr>
<td>Intrusion prevention</td>
</tr>
<tr>
<td>Protection against malware</td>
</tr>
</tbody>
</table>

*Table 1. Security features.*
For each security feature a set of security requirements is defined, describing what needs to be fulfilled in order to fulfill the security feature. An example of a security requirement, taken from the security feature called Security logging in the KSF, is:

*The security function shall make sure that only authorized administrators can maintain the security function and handle its security settings. (Swedish Armed Forces, 2004)*

In NTE a set of security features, with its related requirements, is called a Requirement Collection. The structure of a requirement collection can be seen in figure 9.

![Figure 9. The structure of requirement collections.](image)

Although the requirement collections should be sufficient to describe the security properties of an IT system, another set of requirements is included in the predefined input. These requirements are called Filter functional requirements and describe the different kinds of functionality in traffic mediators. The filter functional requirements used in NTE, as listed in table 2, were specified by Hallberg et al (2006).

<table>
<thead>
<tr>
<th>Filter functional requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Packet filtering</td>
</tr>
<tr>
<td>Stateful-inspection</td>
</tr>
<tr>
<td>Application layer gateway</td>
</tr>
<tr>
<td>Circuit level gateway</td>
</tr>
<tr>
<td>Network address translation</td>
</tr>
</tbody>
</table>

*Table 2. Filter functional requirements.*
4 Implementation of NTE

This chapter describes the implementation of the design described in the previous chapter. The first two requirements in the design express the overall aim for the implementation, while requirements three to eighteen explain the desired functionality. A reference to each of these requirements can be found in each of the following sections.

4.1 Plugin Handling

Requirement 3

A tool or database plugin for NTE is compiled as a Dynamic Link Library, DLL, and placed in a subfolder of the application root folder. The files in this subfolder are traversed at the startup of the application, where NTE loads the assembly from each DLL-file and examines the classes defined inside. If any of these classes implements either the tool or database plugin interface, the assembly is stored in a dictionary indexed by the name and version of the plugin. The plugin resides in this dictionary until it is needed, e.g. when the user performs an operation which depends on that specific plugin. Upon such an event, the application looks up the assembly in the dictionary and creates an instance of the class implementing the interface. This instance is stored in a local variable as the active plugin of its kind.

This solution, where the application keeps the assemblies stored for later use and only initiates them when needed, becomes especially useful in operations like the export method. While exporting, two instances of database plugins are active in order to read from one of them and write to the other. Another benefit of this technique is that a tool plugin can be temporarily initiated to, for example, delete a system.
4.2 Tool Plugins

Requirement 4

The structure of a tool plugin in NTE is not limited in any way other than that it should extend the UserControl class and implement the tool plugin interface, described in the next section. The UserControl class is the base class for user defined graphical components. Extending this class enables NTE to dock the plugin into its main workspace area.

To further integrate the plugin into NTE, it is also given access to the main menu and status strip of the application, via two functions passed on with the initialization call of the tool plugin.

The Tool Plugin Interface

The interface, as shown in figure 10, is used for communication between NTE and the tool plugin and must be implemented in order for the plugin to be recognized by NTE. The interface contains the functions called when the user performs an operation on a system, such as opening or saving. It also contains a method for accessing the active system of the tool plugin.

```
<table>
<thead>
<tr>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ Initialize ()</td>
</tr>
<tr>
<td>+ NewSystem ()</td>
</tr>
<tr>
<td>+ OpenSystem ()</td>
</tr>
<tr>
<td>+ Export ()</td>
</tr>
<tr>
<td>+ CloseSystem ()</td>
</tr>
<tr>
<td>+ DeleteSystem ()</td>
</tr>
<tr>
<td>+ GetSystemCopy ()</td>
</tr>
<tr>
<td>+ SwitchReqCollection ()</td>
</tr>
<tr>
<td>+ ActiveSystem</td>
</tr>
</tbody>
</table>
```

Figure 10. Class diagram of the tool plugin interface.

4.3 Database Plugins

Requirements 5-8

In the design chapter a need for a data access layer that operates on objects was described. In NTE this layer consists of a database plugin that implements a specified interface. NTE and the tool plugin have different needs regarding the needed database functionality. To distinguish the differences in needs, the interface to the database plugin is divided into two parts, one for the tool plugin and one for NTE, as can be seen in figure 11. One requirement from the database plugin is the existence of an interface to recognize classes defined by the tool plugins. This requirement is fulfilled by introducing the NTE database class interface, called NTEDbClass. The interface is empty since its only purpose is to allow the database plugin to recognize the tool plugin classes. Basically, it can be seen as a flag for marking these classes. Marking the classes of the tool plugin as NTE database classes ensures that the classes will preserve their mutual consistency.
NTE Database Plugin Interface

The NTEDbPlugin interface is an empty interface which itself implements the two interfaces, used by NTE and the tool plugin, for communicating with the database plugin. A database plugin must implement this interface in order to be recognizable by NTE.

![Class diagram of the database interfaces.](image)

### Tool Database Interface

The database interface used by tool plugins is designed with simplicity in mind. It contains three methods corresponding to the basic operations on objects in a database: store, fetch and delete.

The store method is defined as a function taking an arbitrary object as parameter. Objects that do not implement the NTE Database Class interface will be serialized and stored as a byte sequence in the database.

The method for fetching objects is a generic nullary function with a parameter T, where T is the type of the objects that should be fetched from the database. The use of a generic parameter in the function call has the benefit of enabling the function to return a generic list with the same element type in order to avoid having to cast the result into the correct type. This also makes the code clearer since it is easy to see exactly what type is involved in the operation. The interface also makes a constraint that the type T should implement the NTE Database Class interface, thus enabling the type checking to be performed at compile time.

The method for deleting objects from the database is a function taking the object to delete as a parameter.

### NTE Database Interface

The interface for accessing the database from NTE is closer to the database implementation than the tool database interface. For NTE to be able to handle systems, two new methods are defined in the interface for fetching and storing systems, both operating on IDs. These have the same functionality as the fetch and store methods in the tool database interface, except for the fetch method taking an ID as a parameter and the store method returning an ID. The interface also defines
an attribute for retrieving the ID of the active system and a method for retrieving the ID of an arbitrary object.

Additionally, the method for initializing the database plugin is defined as a function taking three arguments: the path to the working directory of the plugin, a set of files in which it can find the active database and the system ID to work on. The set of files can also later be retrieved by NTE in order to support the option for the plugin to create new files during its execution.

### 4.4 Common Library

**Requirement 9**

In the design chapter, the common library is described as a collection of components that should assist the developers while designing user interfaces. To be able to assist the developer, it is important that the components in the common library are as easy to work with as any other components. Therefore the components in the common library can be used directly in the design view of Microsoft Visual Studio, which gives the possibility to edit the attributes of the components directly in the design view. It is also easier to design user interfaces with the possibility to place out the components directly in the design view. Thereby the components in the common library appear and behave just like any of the components available in the .NET platform.

The common library contains a variety of components ranging from a textbox with special functionality to handle numeric values, to a form for making AHP prioritizations (Saaty, 1994). It is possible to extend the common library in the future with more components as new needs arise. For a more detailed description of the content of the library, see appendix B.

### 4.5 Project Handling

**Requirements 10-13**

In NTE, a project is a file containing a collection of systems using the same tool plugin and the same database plugin. The systems within a project share requirement collections and also have the possibility to share information specific to the selected tool plugin. Thus, the project can be seen as a database of components and the systems within as different configurations of these components. This enables the user to experiment with different configurations of the same components.

#### 4.5.1 Structure of the NTE File Format

The NTE file format is basically an encapsulation of the project database. The structure of the database and the information it contains is only accessible by using a specific database plugin in combination with a specific tool plugin. Hence, NTE has no knowledge of what is actually stored in the database without invoking the related plugins. In order to still let NTE keep track of the systems stored within a project without having to call the plugins each time the file is accessed, additional
information is stored inside the file. The structure of the NTE file classes are shown in figure 12.

By storing a list of system information, SysInfo, in the NTE file, the systems residing within the project can be listed without invoking the plugins. Each SysInfo instance contains the name of the system, a description, the time of creation, the time of the last modification, a screenshot and the requirement collection currently in use. A SysInfo also contains an ID in order to associate it with a system within the actual database.

NTE needs to be able to determine whether or not the plugins used by a project are available. Without the correct version of plugins available, it is not possible to open systems from a project. Therefore the name and version of the used database plugin and tool plugin is stored in the NTE file, as an instance of the PluginInfo class.

Finally, the actual database files are located within the NTE file. They are stored as objects of the class DbFile, which contains the two data members filename and data. Upon the opening of a project file, each object is copied into the working directory of the plugin as a file with the given filename containing the given data.

![Class diagram of the NTE file classes.](image)

### 4.5.2 Operations

As defined in the use cases in the design chapter the project management in NTE consists of five operations, explained below. All these operations operate on systems.

**Create System**

The assignment of the create system operation in NTE is not essentially to create a system, but rather to create the prerequisites needed by the tool plugin to create a new system. The first part of creating these prerequisites is to collect information from the user about the system to create, such as a name and description. The destination of the new system can either be an existing project or a new one. If the
user chooses to create the system in a new project, the option is given to select a tool plugin, database plugin and requirement collection to use in the new project.

The second part of creating the prerequisites is to take the gathered information and use it to initialize the plugins, as can be seen in figure 13. If the system is to be created in an existing project, the database files need to be extracted from the project file before the database plugin can be initiated.

If the system is to be created in a new project, the requirement collection selected by the user is saved in the database before it is passed on to the tool plugin. If the system on the other hand is to be created in an existing project, that project already contains at least one requirement collection. Since the user easily can switch requirement collection, an arbitrary collection in the database is fetched and then passed on to the tool plugin.

When the tool plugin has been both initialized and given a requirement collection to work with, the prerequisites needed by the tool plugin for creating a new system have been provided.

![Sequence diagram of the create system operation.](image)

**Open System**

The task of the open operation is basically to extract the requested system and the available requirement collections from the project file and pass on the control to the tool plugin. The sequence diagram of the open system operation is shown in figure 14.
Each system resides within the database files of the project it belongs to. Hence the first step of opening a system is to extract the database files from the project file. The files are read from the project file and placed in the working folder, designated to the database plugin.

When the database files are in place, it is possible to initialize both the database plugin and the tool plugin. Since the tool plugin is unaware of any IDs in the database, NTE takes care of the actual fetching of the system. This is handled by reading the ID from the system information in the project file. The active requirement collection is also fetched from the database. The last step of the open operation is to pass on the system and the active requirement collection to the tool plugin.

**Save System**

The database plugin operates on the database files in its designated working folder. Hence, all changes made to the active system affect the extracted database files in the working folder and not the database files stored in the project file. The main task of the save function is therefore to update the database files in the project file. This is shown in figure 15. The save function also needs to update the system information specified in the project file to reflect the changes made in the database.
CHAPTER 4 - IMPLEMENTATION OF NTE

After retrieving the active system from the tool plugin it is stored in the database. The list of database files is then requested from the database and stored in the project file.

Delete System
The deletion of a system is handled by both NTE and the tool plugin. NTE is the only participant able to update the project file to reflect the deletion, while the tool plugin is the only one having knowledge of the related resources to delete. Therefore NTE takes care of deleting the actual system, while the tool plugin deletes all the related resources, as shown in figure 16.

NTE initializes the database plugin and tool plugin before the system to delete is fetched. The system is passed on to the tool plugin which gets the opportunity to delete related resources. After deleting all related resources, the system is sent to the delete function of the database plugin which removes the system from the database. To finalize the deletion, NTE removes the system from the system information in the project file.
Export

The export function is used for exporting complete systems or tool plugin specific resources from one project to another. Resources in the project of the active system can be exported to either a new project or to another existing project. Since the definition of a system is specified in the tool plugin, compatibility can only be guaranteed if the two projects are using the same tool plugin. The export operation is explained in figure 17.

![Sequence diagram of the export operation.](image)

Figure 17. Sequence diagram of the export operation.

To make sure that all predefined input in the source project is also available in the target project, all requirement collections and filter functional requirements are fetched from the source project and stored in the target project.

When exporting a system, not only the data in the database needs to be transferred to the target project. The NTE file also contains information about the systems available in the project and this information needs to be updated during an export. The update of this system information can only be carried out by NTE and therefore it needs to handle the process of exporting systems.

When it comes to shared resources NTE has limited knowledge of the classes defined in the tool plugin. By letting the tool plugin handle this part of the export, the implementer of the tool plugin can give the user the possibility to export any extra resources. At the same time it is a way to see to it that all related resources get exported.
After completing its part of the export, the tool plugin returns the systems that are to be exported to NTE which carries out their export. The export includes both saving the systems in the database and saving the system information in the project file to reflect the new systems added to the project. NTE finalizes the export by storing the new project file on disk.

4.6 Aggregation & Presentation

Requirement 14

Adequate presentation of assessment results, and functionality to aggregate over these results, are vital tools when evaluating security assessment methods and tools. Such functionality lets the user study the effects of a method and observe changes in the output on variations in the input.

In the design chapter, this functionality was to be provided by NTE. It is, however, hard to determine what kind of output a tool produces, and to restrict the output to a certain form would be to limit the possibilities of the tools. Therefore the functionality for aggregation and presentation is handled by the tool plugins. To aid the tool plugin developer, some helpful components have instead been added to the common library. These components can help with the presentation of results and still not limit the developer, since the usage of the component is optional.

The XMASS tool plugin, as described in chapter 6, contains functionality for both aggregation and presentation of results and can be used by tool plugin developers as a guide while implementing this functionality in new tool plugins.

4.7 Handling of Predefined Input

Requirements 15-17

As mentioned in the design chapter the input to the tool plugins in NTE consists of a collection of security requirements, where each requirement belongs to a security feature, and a set of filter functional requirements. The requirement collection is implemented as the three classes described in figure 18. As can be seen in the figure each class has an ID which distinguishes an object from other objects of the same class. These IDs are introduced to allow for comparisons of requirements used by different systems and projects. For instance, if two systems located in different databases are to be merged into a single project, the target database should only contain one edition of each requirement stored in the separate projects, i.e. duplicates should not exist in the new project. Since the objects in a requirement collection are compared at an ID level, as in contrast to an object level comparison, extra functionality is needed in the database layer.
Each requirement collection is stored as a file with the extension .rc. These files are only used in the events where a project is created or a new collection is added to an existing project, upon which the contents of the file is copied into the database. If a matching ID is found in the database, the database is updated with the new version.

NTE contains a requirement collection editor where the user can create and edit collections. This editor is the only option the user has to alter a collection, which gives NTE the ability to control how collections are handled. A collection is considered to be intact if no requirements or features have been added or removed. Thus, if the user only alters names and descriptions the collection is still considered to be the same. Adding the altered collection to a project containing the original collection will only update the names and descriptions instead of storing the collection as a new one, which is useful when for example correcting spelling errors, etc. If a collection, on the other hand, is not intact, new IDs are generated for all features, requirements and also for the collection itself. This results in the altered collection being seen as a completely new collection.

NTE also provides functionality for switching between different requirement collections. All requirement collections available in the database appear in the main menu, where the active collection is selected. When a switch of active collection occurs, the `SwitchReqCollection` method of the tool plugin is called, to notify the tool plugin of the change.

The second part of the predefined input is the set of filter functional requirements, represented by the class shown in figure 19. As described in the design chapter, the set of filter functional requirements are fixed, and is thus statically implemented into NTE. Thereby the requirement collection editor has no support for editing the set of filter function requirements.
5 Implementation of the Database Plugin

This chapter describes the implementation of the database plugin used by NTE. Subjects handled in the chapter includes the structure of the database and its functionality.

5.1 Plugin Structure

In the design of NTE, requirement 6 specifies that the data access layer should operate on objects. This requirement is derived from the need of the developer to be able to effortlessly store and retrieve data. There is however no requirement on the actual database representation of the data, and it is thus up to the developer of the database plugin to choose a suitable representation. In the implementation of the database plugin described in this chapter, a relational database is used. The database is run by the SQLite database engine; a small C library that implements a self-contained, embeddable, zero-configuration SQL database engine (SQLite, 2007).

The plugin consists mainly of three layers, as can be seen from figure 20. The layer closest to the database interface is the Object Cache, which is the functionality for ensuring object consistency. At the bottom is the SQLite engine, which performs the queries on the database. Between these two layers is the Object-Relational Mapping layer, which takes care of the conversion between objects and SQL. The object cache and the object-relational mapping are described further down in this section.
Object/Relational Impedance Mismatch

The fact that the database interface of NTE demands that each database plugin is able to operate on objects introduces a problem, since the classes of the object model must somehow be translated into tables in the relational model. This problem is illustrated by Peak & Heudecker (2006) who state that relational databases manage data while object-oriented applications are designed to model a business problem. They claim that these models have radically different purposes and that it can be challenging to resolve these differences, which they refer to as the Object/Relational impedance mismatch.

The O/R impedance mismatch is a well known problem when combining object oriented programming languages with relational databases. Ambler (2006) states that the reason the relational paradigm and the object oriented paradigm do not work together seamlessly is that the first is based on proven mathematical principles while the latter is based on proven software engineering principles. Ambler continues with saying that in the object paradigm, objects are traversed via their relationship whereas in the relational paradigm data rows of tables are joined. These differences in associations are, however, said by Peak & Heudecker (2006) to be the easiest portions of the mismatch to overcome, since both models support the notion of association. The associations in the object model are simply interpreted as foreign keys referencing primary keys stored in another table in the relational model.

Another important part of the O/R impedance mismatch is in regard to identity. In the object model two objects can be either identical or equal. Two identical objects are in fact the same object, meaning that they refer to the same data in memory. Equality, on the other hand, means that the data of the objects are the same. Peak & Heudecker (2006) say that these different notions do not exist in the relational model, where instead rows in a relational database are only identified by the values they contain. They point to the problem of how to identify objects with their relational counterparts.
The O/R impedance mismatch also includes the problem of type incompatibility and the conversion of types between the object and the database representation. This is a minor problem for primitive types but gets increasingly difficult for more complex structures. An object may contain a composite variable consisting of arbitrary sub types, while the relational counterpart must be sliced up into different fields or even tables.

**Object-Relational Mapping**

The solution to the O/R impedance mismatch is often referred to as the Object-Relational Mapping, ORM, and is the process of storing objects in a relational database. ORM forms the layer between applications and databases, taking care of the conversion of objects to and from the relational format. This includes for example the task of constructing queries to the database. There exists quite a few ORM solutions on the market, with varying requirements on the implementation.

Since the idea of NTE is to minimize the amount of extra work for the programmer, the mapping from the object model to the relational model in the database plugin is transparent to the programmer. The database plugin solves this using data type reflection, a feature of .NET giving the program access to information on the structure of classes at runtime. This feature enables the program to examine an object and extract both the type and data of each member of the object, without knowing its representation beforehand.

**Object Caching**

The structure of the database plugin interface, where only instances of objects are passed between the tool plugin and the database plugin, has two consequences related to the identity problem of the O/R impedance mismatch.

The first is that each fetch operation for a specific entry in the database must always return a reference to the same instance of the object represented by the entry. If this problem is not handled the tool plugin will receive clones of the object which will break the consistency of the object structure. An example of this situation is when there are two instances of the same class, A₁ and A₂, where both instances hold a reference to the same instance of the class B. Any changes to the B member of A₁ must also change the B member of A₂. Likewise, a comparison of the B members must evaluate to true.

The second problem is that each store operation on a specific object must operate on the same entry in the database. More explicitly, if an object is stored, which is already present in the database, the entry should be updated with the data of the object instead of being inserted as a new entry.

These two problems are both solved with the introduction of the Object Cache. The main objective of the object cache is to maintain a table of previously handled objects in the database. Each element in the cache is a triple of Type × ID × Object, where Type is the type of the object, ID is the identity of the entry in the database and Object is the actual object. The functionality of the object cache is both
Type \times ID \rightarrow Object, in order to look up an object given its ID, and Object \rightarrow ID to retrieve the ID of an object.

During a fetch operation the program will check the cache for a reference to an instance of the given type and ID. If a match is found, the program will return the cached object instead of fetching it from the database thus speeding up the process and maintaining the consistency between object relations.

During a store operation the program will check if the cache holds a reference to the given instance. If so, the ID in the cache will be used for updating the object in the database, in contrast to storing it as a new entry.

Section 4.7 points out the need for database support of the special handling of requirement collections. It is stated that objects belonging to a requirement collection should be compared by ID in order to distinguish them from each other. This is resolved by keeping a dictionary of discovered objects, mapping their ID to the ID of their entry in the database. In each operation on the database the object is checked to see if its type is any of the three defined classes.

5.2 Table Structure

Since no information about the classes in the different tool plugins are known at compile time, tables have to be created dynamically at runtime. The database plugin maintains a dictionary of discovered data types, storing the necessary information needed to operate on objects of those classes. If a new data type is discovered in a store or fetch operation a method is called to build the table structure for the new type. Each public data member of the class is then examined and based on the data type a suitable database representation is chosen. Only classes implementing the NTE Database Class interface, as described in section 4.3, are taken into consideration.

For primitive data types the process is trivial; a string is stored as text, an integer as integer, a floating point number as real, etc. This is illustrated in figure 21.

![Figure 21. A simple class structure and its corresponding table structure](image)

However, if the program discovers another NTE Database Class, the data member is seen as a one-to-many relation, with the data stored in a separate table and the ID of that table row stored in the current table, as illustrated in figure 22.
The plugin also recognizes generic lists, which are interpreted as a many-to-many relation between the current table and the table of the class of the list element. This results in an intermediate table named $ListOfX$ being created where $X$ is the name of the element class, as shown in figure 23. $ListOfX$ contains two values: an ID relating to the original table and an ID relating to the table of the element. If the element is a primitive data type, the data is simply stored as a value in $ListOfX$.

The same procedure used for generic lists is used for generic dictionaries, with the exception that the relation contains three fields: the ID relating it to the original class, a field $key$ referring to the key of the dictionary and a field $value$ referring to the value of the given key in the dictionary. In this situation, $key$ can be either a primitive data type or an ID of a related table or even another generic list or dictionary. The name of the resulting table is $DicOfXAndY$, where $X$ and $Y$ are the names of the data types for the key and value of the dictionary. An example is displayed in figure 24.
CHAPTER 5 - IMPLEMENTATION OF THE DATABASE PLUGIN

The naming scheme of relation tables where the table is named after its content has the benefit of keeping down the number of tables in the database, since multiple classes can use the same relation table.

Complex data types which do not implement the NTE Database Class interface such as for example a two dimensional array of arbitrary size containing integers, are simply serialized and stored as a byte sequence in the database.

The information obtained from the discovery process described above is also stored in memory as a dictionary relating class names to their corresponding table structure. This information is used in all subsequent operations on objects of the given class to avoid having to examine the data type each time.

5.3 Operations on the Database

As stated in the database plugin interface, described in section 4.3, there are three operations that can be performed on the database: fetch, store and delete.

Fetch

The first step in a fetch operation is to look up the table information for the requested class. The next step is to construct an SQL query for selecting the relevant data from the database. If the class has a simple structure the query is trivial, but if the structure of the class is more complex, the query contains join clauses in order to resolve any associations. Sending the query to the database results in a set of matching rows. For each row of the result, the ID of the row is checked against the object cache to see if an instance of the given class and ID already exists. If an ID does not exist in the cache, a new instance of the class is created by calling the constructor of the class with the values retrieved from the database. These instances are merged with the one already residing in the cache into a list which is returned to the calling tool plugin. Any newly created instances are added to the cache for future use.

Store

When storing an object in the database the table information for that class is looked up in the dictionary. An SQL query is constructed by reading the value from the given object for each field in the table information. Depending on whether the object is present in the object cache, the query will be either an insert or an update statement.

Any related lists or dictionaries are stored using a recursive function call which stores the related data and returns the ID of the data in the related table. Special care has to be taken when updating a relation; since the content of a list or dictionary can and most likely will have changed since it was previously stored, all previous relations must first be deleted. This is solved by a recursive function call traversing the table information tree and deleting all related data that is not an NTE Database Class from the database.
Delete
When the tool plugin wants to delete an entry in the database it calls the delete function, passing along a reference to the object to delete as an argument. The table ID of the given object is then determined by looking it up in the object cache. If the given object is not present in the object cache it is neither fetched from nor stored in the database and thus the function call is ignored. If an entry is found however, the entry is deleted as well as the entry in the object cache.
6 Implementation of XMASS

This chapter contains a description of the design and implementation of the tool plugin based on XMASS. For a closer view of what the actual implementation looks like, the reader is directed to the user guide in appendix A.

6.1 Design Focus

This tool plugin implementation will mainly be used to evaluate the method. Therefore the focus has been on creating an implementation where values and settings easily can be varied. By facilitating the variation of values, it is possible to see how alterations of the input affect the assessment result. Given the knowledge of how a certain variation should affect the security it may be possible to draw conclusions from the assessment result about the soundness of the method.

6.2 Graphical User Interface

The development of assessment methods is, as previously stated, still in an early state where the implemented tools are mainly used by researchers for evaluating and comparing assessment methods. The target group of this tool plugin implementation is assumed to be familiar with the area of security assessment in general and to have good knowledge about XMASS and thereby recognize most of its used notation.

When designing a graphical user interface (GUI) it is important to build it upon the existing knowledge of the user. By using familiar concepts in the GUI, the user learns quicker how to use the application. This familiarity can be achieved by for example using language and expressions that are familiar to the user. (Galitz, 2007).

The level of familiarity can also be increased by designing the GUI in a way similar to other applications in the same genre which the user has been working with. The GUI design of the XMASS tool is based on the design used in the MASS implementation.
known as ROME2. Placing the different parts of the GUI in a similar way as in ROME2 leads to an easier orientation for the user.

The GUI should be designed in a way that results in the user knowing what to do and where to look. One important step in achieving this obviousness of where different settings are to be found is to group the settings in a logical way. One example is the gathering of all settings that affects the whole system into the same dialog. By grouping the system related settings, the user also gets an overview of the settings that can be made for a system.

### 6.3 Method Improvements

In this implementation some improvements are made to XMASS. Two new profiles are introduced along with an extension of traffic generating entities.

**Profiles**

As described in section 2.4, XMASS is using profiles to model the security values and filtering values of entities. A profile can be seen as a grouping of values which concerns one or more entities. This structure gives good possibilities to facilitate the task of switching between different groups of values.

This implementation introduces two new profiles, the physical relation profile and the logical relation profile. The introduction of these two new profiles does not alter the method itself, but it groups the values of a relation and gives the option to easier switch back and forth between different groups of values. Both relation profiles are implicitly described by Hallberg et al (2006), but neither is referred to as a profile. A similar implementation of a logical relation was made in the assessment tool ROME2, but neither here referred to as a profile.

The physical relation profile is set system-wide and is used to model all physical relations in the system, while the logical relation profile is set per logical relation. By setting the logical relation profile per relation, all logical relations could be using different profiles. Thereby it is possible to model different kinds of logical relations like for example VPN (Virtual Private Network) tunnels. One positive side effect of this introduction is that the consistency of the implemented method is increased by also having the values of a relation grouped together as a profile.

**Quantity**

The traffic generators described in XMASS (Hallberg et al, 2006) have been extended with a quantity value. This value specifies how many equivalent traffic generators an entity represents. As a result of this introduction, all traffic generators having a quantity larger than 1 are considered being a network of entities. The quantity feature can be used to model either a stack of equivalent entities or a public network.

### 6.4 Result Aggregation and Presentation

One of the goals of the work presented in this thesis was to implement some form of aggregation over the results from an assessment. The aggregated result was then to
be presented to allow conclusions to be drawn about the impact of the changes in the system. This functionality was first planned to be provided by NTE, as described in chapter 4, but after some consideration it was believed to limit the developer’s possibilities to present assessment results and thereby reduce the user’s possibilities to evaluate the method. Therefore this functionality needs to be implemented in the XMASS tool plugin.

One suggested idea to provide this functionality was to let the user manually vary the values and give the possibility to save each assessment result. The user should then be given the possibility to compare these saved assessment results.

In the XMASS implementation this is solved in a more sophisticated way, as can be seen in figure 25. The user is able to define start and end states of the values in the system. These states are used to create a series of intermediate states using linear progression from the start to the end state. Each state is assessed using the same calculations as in the normal system assessment, but instead of displaying the result it is stored in a vector. This vector is sent to a presentation view which displays the result as graphs for the different components of the system. This solution enables the user to perform evaluations on the system without altering the original state.

![Diagram](image)

**Figure 25.** Result aggregation and presentation in XMASS
7 Conclusions

This chapter contains a discussion regarding the work presented in the previous chapters. The fulfillment of the goals, set up in chapter 1, is discussed as well as what conclusions can be made based on the results. Additionally, some suggestions for future development are presented.

7.1 Discussion

The previous chapters describe the design and implementation of an environment with the aim to aid future development in the area of system security assessment. The environment, called NTE, is designed to host various security assessment tools as plugins.

In the design chapter, two actors are identified with different perspectives on NTE – the developer and the user. Although these actors may in fact be the same physical person, both perspectives are used in the design of NTE. In this section, the same actors are used as a base for the discussion regarding the fulfillment of the aim of NTE.

7.1.1 Developer Perspective

In the design chapter, the task of implementing tools realizing security assessment methods is identified as a demanding process, where a large part of the implementation is spent on basic tasks like file handling and user interface solutions. One of the aims of NTE is to assist the developer by reducing the time needed to be spent on these tasks, while at the same time avoiding any limitations on the tool implementation. A developer of security assessment tools is predicted to have three main advantages of utilizing NTE.

The first advantage is the use of the data access layer, which provides the developer with the functionality to store, fetch and delete objects in a database. The layer is designed to be transparent to the developer and provide simple operations for
handling the contents of the database. The developer can work solely on objects and
does not have to modify the code or the structure of the classes to fit the database.
There might be one drawback with this solution, though. Since the developer does
not accommodate the objects to the database, the database plugin itself has to take
care of this. This may have an impact on the performance of the database operations,
since the database plugin has to discover the structure of each object in run-time.
Although this might be a problem in an end-user application, the benefits out-
weights the disadvantages in this situation.

NTE also provides the developer with a well defined interface for tool plugins,
which facilitates the development and works as a guide for determining the control
flow of the program. It should be easier for the developer to write code for each
procedure defined in the interface, in contrast to the process of identifying the
needed functionality while implementing.

When it comes to the GUI programming, NTE assists the developer by providing
the components available in the Common Library, which can be used directly in the
design view of Microsoft Visual Studio. Thereby the developer has free hands to
design the GUI, but with the option to use the provided components as it fits. It
should be pointed out though, while the Common Library can assist the developer
during the implementation of the GUI, the GUI programming is still a time
consuming task when implementing a security assessment method. Even so, the
developer should be able to save time by utilizing the Common Library.

While providing the developer with helpful features, there are also some drawbacks
compared to the ideal environment described in the first chapter. The predefined
input is designed to fit the KSF, which might limit the generality of NTE.
Additionally, even though the aim of the design is to make NTE general enough to
fit for the implementation of most security assessment methods, XMASS is the only
security assessment method implemented in NTE. This rises the possibility that the
environment is too adapted for XMASS and thereby not as general as intended. It
should, however, be at least general enough to support future implementations of
security assessment methods developed at FOI.

7.1.2 User Perspective
From the user perspective, NTE offers a range of advantages which assists the user
while evaluating or comparing methods. The project manager gives the user an easy
way of handling projects, which helps organizing similar systems and provides the
feature of letting different systems share resources. The ambition of the project
manager is to inspire the user to experiment with different configurations and
thereby have better possibilities to evaluate methods.

Since security assessment methods are implemented in NTE as plugins, the user can
assess systems using different methods within the same application. Together with
the requirement that all tool plugins have to use the predefined input, different
security assessment methods can easier be compared.

While the use of predefined input helps comparing different methods, a common
representation of the output data would simplify the comparison. By having both the
same input and the same form of output, it would be easier to compare how
different security assessment methods behave. As concluded in the discussion about
aggregation and presentation in section 4.6, the use of a common output could limit
the possibilities of tool plugins. The consequence is that the functionality to
aggregate over assessment results and the presentation of the results is provided by
each tool plugin instead of NTE. This could limit the possibilities of the user to both
evaluate and compare methods since NTE cannot guarantee the quality or
functionality of the aggregation and presentation.

7.2 Results
In the first chapter the aim for this thesis is presented with the main task to
implement an environment supporting future implementations of tools for security
assessment. In conclusion of the discussion in the previous section, this part of the
aim can be considered as achieved, with some reservations to the aspects of the
generality of NTE. However, as previously concluded, the environment should at
least be able to support future development on XMASS, but only time will tell
whether NTE is general enough.

The design and implementation of a database solution must also be considered a
success. The database is simple to use and might only, as stated earlier, lack when it
comes to performance.

The only part of the aim of this thesis which is not achieved as intended is the
support for aggregation over assessment results and presentation of results which
were designed to be included as a part of NTE. Attempts have been made to include
this functionality in NTE, but as discussed earlier this functionality needs to be
provided by the tool plugins. To aid the developer implementing this functionality,
the Common Library contains components which can be used for presenting the
results.

The last part of the aim of this thesis, to implement XMASS into the environment,
can also be considered as successfully achieved, and should permit for upcoming
evaluations of the method. The implementation also has a positive side effect for
other developers, since the code can easily be reused in other tool plugins and thus
allow for even more time saving in future implementations.

7.3 Suggestions for Future Improvements
Even though NTE already is operational, some suggestions for future improvements
have been found:

When an updated version of a tool plugin is released, the systems created with the
previous version might not be compatible with the new version of the tool. The user
can still work with the old systems as long as NTE has access to the previous version
of the tool plugin, but a conversion tool which updates the old systems to the current
version might be needed to simplify an update for the user, who wouldn’t have to
remodel the old systems in the new version.
As discussed in section 4.6, the functionality to aggregate over the assessment results and present the aggregated result is not handled by NTE. If a suitable representation for a common output data can be identified, this functionality should be added into NTE and thereby reduce the amount of work for the tool plugin developers and at the same time guarantee a certain level of quality and functionality.

The Common Library contains a range of different components that are intended to help the tool plugin developer while implementing the GUI. As more tool plugins are developed, the library should be extended with more components to aid the developer even more with the GUI implementation, which still is the largest part of creating a tool plugin implementation of a security assessment method. Extending the Common Library raises a possible need for version handling of the library as its content might differ. Thereby an extension of the Common Library with more components and version handling functionality is suggested.
References


Appendix A - User Guide for NTE & XMASS

This guide describes how to use NTE with the XMASS plugin to assess the security of a system. The guide is divided into two sections. The first section is about the main program called NTE, and the other section is about the XMASS plugin, which implements the eXtended Method for Assessment of System Security (Hallberg et al, 2006). An example of NTE running XMASS can be seen in figure 26.

![Figure 26. An example system running NTE with XMASS.](image)
NTE

NTE is an application for running tool plugins. A tool plugin contains everything that is specific for the security assessment method, while NTE contains more basic functionality like handling project files and requirement collections.

Requirement Collection Editor

The main input to the tool plugins are requirement collections consisting of sets of security features, where each security feature has a set of security requirements. Requirement collections are created and edited in the Requirement Collection Editor, accessed by selecting it from the Requirement Collections menu. Each collection is saved in a separate file and can later on be imported into a project. By default a version of the KSF requirement collection, assembled by the Swedish armed forces (2004), is included with NTE.

Creating a New Requirement Collection

A new requirement collection is created by selecting New from the File menu in the Requirement Collection Editor. This will bring up an empty collection. Since the requirements are grouped by security features the first task is to add a new security feature by clicking on the New button under the security features drop-down list. This will highlight the security feature group box, as can be seen in figure 27.

![Requirement Collection Editor](image)

Figure 27. Adding a new security feature for the collection.

After giving it a name, abbreviation and an optional description the security feature is created by clicking on the Apply button. Security requirements can now be created
for the selected security feature by clicking on the New button under the empty security requirements list on the left side of the dialog window. After giving it a name and an optional description the new requirement is created by clicking the Apply button.

By repeating the steps above, all features and requirements needed may be added. When the collection is complete it can be saved by selecting Save as from the File menu. The dialog displayed in figure 28 will appear, asking for a name for the collection. Existing collections can be overwritten by selecting them from the list.

![Figure 28. Saving a collection.](image)

Editing an Existing Requirement Collection

An existing requirement collection is opened by selecting Open from the File menu in the Requirement Collection Editor. A dialog will appear showing the existing collection files. Selecting a collection from the list and clicking the Ok button will allow for the collection to be edited. A requirement or feature is changed by selecting it and clicking the respective Edit button. It is also possible to delete requirements or features by selecting the one that should be deleted and clicking the respective Delete button.

If the changes made are only minor, such as editing the name or description of a feature or requirement, the collection can be saved under its old name. However, if the changes are major, such as adding or deleting a feature or requirement, the collection must be saved as a new collection.

Projects

The idea with projects is to group systems using the same components. The components created in one system will also be accessible from other existing systems and new systems within the same project. A project can only use one tool plugin and one database plugin, which are selected while creating a new project. To be able to
assess a system using another tool plugin, a new project using the desired tool plugin has to be created.

Tool Plugin
A tool plugin is an implementation of an assessment method used for modeling and assessing system security. NTE can have a varying set of tool plugins, but only one tool plugin can be used in each project file. XMASS is an example of a tool plugin.

Database Plugin
It is possible to save projects using different kinds of database representations. DbSQLite is an example of a database plugin.

Systems
A system contains all settings and possibly a visualization related to what to assess. Exactly what a system contains depends on the tool plugin in use.

Creating a New System
A new system is created by clicking on New in the File menu, which will bring up the dialog shown in figure 29. Start with entering a name and description for the new system. The next step is to choose whether to create the system in an existing project or in a new project. Systems can be created either in an existing project, by selecting the radio button marked Existing project and choosing the existing project from the drop-down list, or in a new project, by selecting the radio button marked New project. The latter requires a selection of Tool and Database plugin and a Requirement collection to use in the new project file.

Figure 29. Creating a new system.
Switching the Active Requirement Collection

It is only possible to have one requirement collection active at a time. To assess a system using another collection the active collection has to be changed, by selecting *Active collection* from the *Requirement Collections* menu. A list of all collections available in the project will be displayed, as seen in figure 30, with the active collection marked with a checkbox next to the collection name. More collections can be added to the project by selecting *Add/Update*. If the selected collection already is present in the project, it will be updated.

![Figure 30. Switching the active requirement collection.](image)

Opening an Existing System

An existing system can be opened by selecting *Open* from the *File* menu. In the dialog that appears is a list of all existing projects and their respective systems. By selecting a system it is possible to see a screenshot of that system. It is also possible to see the versions of the tool and database plugins used in the project containing the selected system. If the right versions of the needed plugins are available the *Open* button will become active after selecting a system.

Deleting an Existing System

In the *Open System* dialog it is also possible to delete systems, by selecting the system to delete and clicking on the *Delete* button. It is important that the right system is selected since it is not possible to undo this action.

Saving a System

When working in a new system, the only save option available is to select *Save* in the *File* menu. This will save the current system using the system name entered when creating the system. If the new system is created in a new project, the program will ask for a name to use for the project file before the system will be saved. If the system has been saved earlier it will also be possible to save the active system as a new system. This is performed by selecting *Save copy* from the *File* menu. The system will be saved as a new system in the same project and the work will continue in the created copy.

Exporting Systems

An open system can be exported from the active project to either a new project or another existing project. This is performed by selecting *Export* from the *File* menu. A dialog will appear, resembling that of figure 31, asking for which project to export the system to.
The XMASS Tool Plugin

By using the XMASS plugin it is possible to assess the security of a system based on the eXtended Method for Assessment of System Security, XMASS (Hallberg et al, 2006). XMASS is based on the assumption that the security relevant qualities of entities can be described as a set of security features with associated elementary security values. These elemental security values are referred to as security profiles.

System Settings

To be able to assess a system using XMASS, some system-wide settings have to be set. These settings are found in the System Settings dialog, depicted in figure 32, which is accessed by selecting System Settings in the XMASS menu. The first step in this method is to create the templates needed for computing profiles.
There are two types of templates in XMASS – security profile templates and filter profile templates, used for computing security profiles and filter profiles, respectively. Both types of templates are created using the method for criteria weighting from the Analytical Hierarchy Process, AHP (Saaty, 1994).

Security Profile Templates

A system needs to have a security profile template, since the selected template will be used system-wide for creating the security profiles used by the entities. The requirements classed as Important will be used in the prioritization, while the ones classed as Fundamental will be multiplied with the result of the prioritization.

Security profile templates are created by navigating to the System Settings dialog and clicking on the New button under the security profile template drop-down list. The dialog that appears, as seen in figure 33, has one tab for each security feature in the active requirement collection.

In each tab the requirements need to be divided into two groups; Important and Fundamental. If there are two or more requirements in the important list, the AHP button will become active, and by clicking on it is possible to prioritize the important requirements.

In the top right corner of the dialog it is possible to set how much the important requirements should influence on the security value. The entered value should be in
the interval $[0, 1]$. A security profile template requires at least one important or fundamental requirement for each security feature.

![Security Profile Template](image)

**Figure 33.** Creating a security profile template.

**Filter Profile Templates**

To be able to have a system with traffic mediators a filter profile template has to be selected. This template will be used system-wide for creating filter profiles and is used to map the security functionality of traffic mediators to the security features represented by the security profiles.

Filter profile templates are created in the *System Settings* dialog. A new template can be created by clicking on the *New* button under the filter profile template drop-down list. The dialog, as seen in figure 34, has one tab for each security feature in the active requirement collection.

The functional requirements can be prioritized for each security feature by clicking the *AHP* button. There is also a Filtering influence factor that can be set for each security feature. The filtering influence factor is a value in the interval $[0, 1]$ deciding the maximum value of the filter profile value.
**System Functions**

The system function settings can be found in the upper left corner of the dialog. A system function needs to be set for each security feature in the active requirement collection. The system functions are used in the calculation of the system-dependent security profile.

**Effective Filter Profile Weights**

The EFP weights can be found in the upper right corner of the dialog. These weights define how much the security profile affects the filter profile for each security feature. Each weight is a value in the interval \([0, 1]\). The weights are required to sum up to 1.

**Profile Manager**

Profiles are used to model how an entity or relation affects the security of the system. There are four kinds of profiles – security profiles, filter profiles, physical relation profiles and logical relation profiles.
Security Profiles

Security profiles are used to model the security values of all entities, with one security value for each security feature. Security profiles are created in the profile manager, by clicking on the tab marked Security profile and then clicking on the New button. This will produce the dialog depicted in figure 36. Start with giving the new security profile a name and description. Then enter the security values for the important and fundamental requirements corresponding to the security profile template selected for the system. The security values should be in the interval [0, 1].
Filter Profiles
The filter profile describes the ability of a mediator to filter malicious traffic. This ability is represented with a filtering capability value for each filter functional requirement. Filter profiles are created by navigating to the Profile Manager dialog and clicking on the tab marked Filter Profile and then on the New button. This will bring up the Filter Profile dialog, as can be seen in figure 37. The dialog contains a list of all filter functional requirements, where each requirement must have a filtering capability value in the interval [0, 1].

Physical Relation Profiles
The physical relation profile is global for all physical relations in the system. Since it is a global setting for the whole system, it can be found in the System Settings dialog. The physical relation profile can be found at the bottom of the dialog. A new
physical relation profile can be created by clicking on the New button under the physical relation profile drop-down list.

The dialog that appears, depicted in figure 38, has a matrix where the relative weight and function can be set for each security feature in the active requirement collection. The relative weights must not sum up to more than 1 for each row in the matrix. The weights and functions will be used for calculating the neighbor-dependent security profiles.

Logical Relation Profiles
The logical relation profile is set for each logical relation in the system. Hence all logical relations could use different profiles. Logical relation profiles are created in the Profile Manager, by clicking on the tab marked Logical relation profile and then clicking the New button to create a new logical relation profile. The matrix, used to specify the effects of the corresponding described relations, is the same as for the physical relation profile described above.

Entities
There are two types of entities – traffic generators and traffic mediators. A computer would for example be classified as a traffic generator, while a firewall would be classified as a traffic mediator. An entity is created by double clicking on the workspace. The appearing dialog has the option to create a traffic generator or a traffic mediator. Whether creating a generator or a mediator, a security profile has to be selected for the new entity, by either choosing an existing security profile or creating a new by clicking the New button.
Traffic Generators
Selecting the radio button marked Generator will activate the fields for weight and quantity. The weight, a value in the interval [0, 1], represents the importance of each entity. A high weight represents an important entity. The weight is used for calculating the system-wide security profile. The quantity defines the number of equivalent generators it should represent. This way it is possible to have one entity that represents, for example, ten equivalent entities. It can also be used to emulate a public network, like for example the Internet. Example of traffic generators are displayed in figure 40.

Traffic Mediators
After selecting the radio button marked Mediator a filter profile can be selected to use for the mediator. Either use an existing filter profile or create a new one by clicking the New button. An example of a traffic mediator is displayed in figure 41.

Relations
There are two types of relations – physical relations and logical relations, as displayed in figure 42. A relation between two entities is created by holding down the right
mouse button on an entity and dragging the mouse to another entity. A dialog will appear where it is possible to choose to create a physical relation or a logical relation, as seen in figure 43.

Figure 42. Example of relations.

Figure 43. Creating a relation.

Security Assessment
A system will automatically be evaluated when all needed settings have been made. The result of the evaluation can be found on the left side of the workspace. The upper part contains the calculated values for the whole system, while the lower part contains values for the selected entity.
Selecting system security assessment from the XMASS menu will start a tool for varying the values of the templates and profiles in the active system. Select a template or profile and click the Set end state button to set the end values. Do this for all the values that should be varied. Click the Evaluate button to start the evaluation. One hundred evaluations will be made starting with the original values of the templates and profiles, and ending with the defined end values.

**Figure 44.** Assessment data
In the assessment results that appear after the evaluation it is possible to see how the value variations affect the system values as well as the system-dependent values of each generator.
Figure 46. Assessment results.
Appendix B - Common Library

This appendix describes some of the major components of the Common Library.

AHP Form

The AHP Form is an implementation of the method for criteria weighting from the Analytical Hierarchy Process, AHP (Saaty, 1994). AHP is a process for decision-making and is based on pair-wise comparisons of criteria. The result of the process is an eigenvector, where the maximum value of the vector corresponds to the most important criterion.

The AHP form is implemented as a generic class which takes a list of objects implementing the AHPClass interface, and a list of values representing the values of the sliders in the form. For each object in the list, a tab is created containing sliders that are used to do a relative prioritization with the other objects in the list. Based on the values from the relative prioritization, a matrix is formed. The priority of an object is received from the calculation of the normalized values of the eigenvector that corresponds to the maximum eigenvalue for the matrix.

![AHP Form](image.png)

**Figure 47.** AHP Form for prioritizing four requirements.
Graph Panel
The Graph Panel is a component for modeling graphs consisting of vertices with unidirectional and bidirectional edges. Upon the creation of a vertex or an edge, an object is requested from the tool plugin. This object is bound to the created vertex or edge and is used for subsequent events. For example, when an edge is created between two vertices, the triggered event will contain the objects bound to the vertices. This has the benefits of letting the developer of a tool plugin ignore the actual graph representation and work solely on the objects of the tool plugin.

![Diagram of a Graph Panel](image)

**Figure 48.** The creation of a new relation in a Graph Panel.

NTEGraph
The NTEGraph component is useful for displaying assessment results. It takes an arbitrary number of vectors containing decimal numbers and displays these as color coded lines. If two or more series have identical values they are drawn as a single dashed line with the colors of the series.
Figure 49. An instance of NTEGraph displaying the changes in the assessment results.
På svenska

Detta dokument hålls tillgängligt på Internet – eller dess framtidiga ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extra-ordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

In English

The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/

© Johan Bengtsson & Peter Brinck