GRADUATE PROJECT:

AUTHORS: Magne Andreassen
Jon Langseth
Stian Ramse

Date: 19.05.05

Summary of Graduate Project

Title:	Copwall	Nr: 6
		Date: 19.05.05
Participants:	Magne Andreassen	
	Jon Langseth	
	Stian Ramse	
Teacher supervisor:	Erik Hjelmås	
Employer:	Espen Torseth representing Gjøvik University College	
Contact person:	Espen Torseth	
Keywords (4)	Copwall, OpenBSD, Firewall, Java	
Pages: 68	Appendixes: 13	Accessibility(open/confidential): Open

Short description of the project:

The Copwall project presents the foundation in form of architecture, design and application prototype of the Copwall Firewall management system.

The Copwall Firewall management system aims to solve the task of configuring, maintaining and monitoring network firewalls, by presenting a powerful graphical application and a centralized management solution. The Copwall application GUI will give the user access to both simple and advanced features of firewall filter configuration, along with system service configuration and management features. The Copwall management system will provide a central management solution for distributed firewalls, using encrypted channels in all network communication.

The Copwall system uses the security of OpenBSD and the power of Packet Filter (PF) to provide a strong and secure solution on firewalls and management server, while allowing platform independence for the user, by implementing the user interface in the cross platform compatible Java system.

This report presents the result of five months work by three graduate students at Gjøvik University College.

Preface

This report is the culmination of roughly five months worth of work. It presents our efforts on the graduate project that all students have to do during the last semester in Bachelor of Computer Science. The graduate project has a workload of 20 ECTS points.

We would like to direct thanks to Erik Hjelmås, who has been our teaching supervisor, and Espen Torseth who has represented our formal employer for this project, Gjøvik University College. They have both contributed with great ideas, advice and technical know-how, and shown great interest in both the project and the resulting end product. Thanks also go out to Geir Ingebrigtsen, who has assisted with external testing and ideas, and a strong belief in the product's viability in the system administrator market.

We also thank the group that has shared room with us, Roger Carson, Cristopher J. Fullu, Halvor Borgen and Kenneth Fladby, for the good working environment these five months, and for their help in testing of our application.

Gjøvik 19.05.05

Magne Andreassen	Jon Langseth	Stian Ramse

Contents

List of Figures

Chapter 1

Introduction

1.1 Problem description, limits and constraints

Anyone connecting a computer to the Internet, or any other large scale network today, are exposed to the risk of network attack, intrusion, viruses, and otherwise malicious network usage. All users of network systems today, both single home computers, for personal use, and large-scale corporate networks, containing sensitive data, depend on a form of first line defence. This first line of defence is set up using a network packet filter, be it a piece of software running on the user's workstation, or a dedicated computer system, running the filtering for an entire network. These packet filters are what is called firewalls [1].

With a single workstation, connected directly to the Internet, a packet filter software application installed on the workstation normally is enough. But for a network of computers, this becomes an inefficient solution, and a dedicated firewall is needed. This can be implemented using purpose-built hardware, but in many cases this becomes either limiting in way of functionality, or expensive, if not both. Using a standard computer, and a suitable combination of operating system and filtering software, provides a far more flexible solution, and may become less expensive. Especially if this is done using a strong and secure operating system like OpenBSD.

The task of configuring, maintaining and monitoring a firewall system or cluster of firewalls through a shell prompt, is time consuming and complex. To help do something with this, a management application with a logical and easy to use graphical user interface is needed. Such an application does not currently exist for OpenBSD. In addition, no solution does currently exist for centralized management of multiple firewalls running OpenBSD or other open-source operating systems.

This project is set to rectify this, by creating the foundation for a software product for graphical configuration and centralized management of firewall systems. This will be done by demonstrating a powerful concept for graphical administration and management of OpenBSD based firewalls, and creating an architecture for

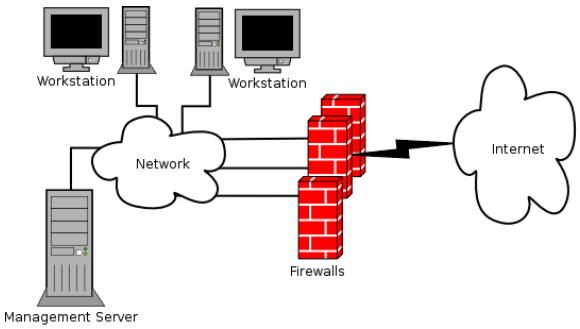

Figure 1.1: A simple example of a Copwall managed network topology.

centralized management of firewalls, and include functions such as revision control and encrypted communication in this architecture.

The final software product shall support both simple and advanced user features in configuration of firewall functionality on OpenBSD, using the PF[1] and CARP[2] features of OpenBSD.

In addition to this, the system will support configuration of running system services, all configuration of network interfaces, and to a more limited extent configuration of operating system options.

The software is set to be built as a three tier architecture, on top the OpenBSD operating system and other essential supporting services, such as database engine or virtual machine. The three tiers of the system will be:

- Graphical Management Client, abbreviated GMC, a graphical user interface that will typically be used on a non-OpenBSD workstation connected to a firewalled network.
- Central Management Platform, abbreviated CMP, a middleware system that can be installed on a separate management server, or directly on a computer acting as firewall
- Configuration and Administration System, abbreviated CAS, to be installed on computers performing firewalling, and handles the actual administration of the firewall computer and/or firewall cluster.

The software system shall support administration of multiple firewalls, and support separation of the concepts:

- Firewall Physical Units, abbreviated FPU, physical computers acting either as standalone firewalls, or as part of a failover firewall cluster

[1] Packet Filter is the filtering engine (firewall) bundled with OpenBSD
[2] Common Address Redundancy Protocol is used by services like PF to gain high availability

- Firewall Logical Units, abbreviated FLU, consisting of either a single FPU performing standalone firewalling, or a collection of FPU's in a firewalling cluster totalling as a single firewall.
- Firewall Logical Groups, abbreviated FLG, a visual representation collection multiple FLU's that are grouped together for simpler visual access by the user.

Although it is a goal to come at close as possible to a complete software system, it is quite evident that the amount of work this implies, cannot be accomplished with good quality within the time frame of the project. The project is therefore limited to creating the architecture needed, and demonstrating the central concepts, through documentation, graphical mock-ups and some working code.

1.2 Project definition

The goals of the project are to:

- demonstrate a different view on managing firewall filtering rules, compared to traditional tables-based GUI management.
- create a prototype application through code and mockups that explores the user friendliness of the GUI concepts.
- evaluate and select encryption methods for secure communication.
- provide the overall architecture of the software system.

1.3 Background for the project

The basic project idea came up during spring of 2004, after OpenBSD 3.5 was released. One of the major new features included in OpenBSD was the CARP technology. CARP was the OpenBSD project's response [2] to Cisco's patented HSRP[3], which causes licensing problems using the IETF VRRP[4]. CARP has several design advantages over Cisco's HSRP, and is free of patent rights.

Combining CARP with PF makes for a truly powerful and potentially very secure firewall failover solution. PF is a stable and complex filtering engine that makes OpenBSD perfect for running as a dedicated firewall. PF has implemented quality of service features directly in the filter syntax, and is capable of doing most things an expensive commercial firewall can do.

[3] Hot Standby Router Protocol
[4] Virtual Router Redundancy Protocol

What sparked the idea of the project was an interview with the developers of PF, published at ONLamp.com, where Ryan McBride stated:

> [...] doing a good GUI configuration tool for PF is very difficult because there are so many options, and laying them out intuitively in a graphical interface is nearly impossible. [3]

In contrast to many commercial solutions, OpenBSD does not provide commercial support. Taking into consideration that many users are perplexed by the command line interface they have to overcome, along with potentially complex configuration format, these users do not choose OpenBSD as a firewall. This project will try to cope with this problem, and hopefully make OpenBSD a more widespread operating system, thus ensuring future existence of the OpenBSD project.

1.4 The background and qualifications of the group members

All three members of the project group started at Gjøvik University College in the fall of 2002, at the Engineer of Computer Sciences study. Magne Andreassen and Jon Langseth have both chosen the specialisation of Software Engineering, while Stian Ramse specialises on System Administration. While Magne has a general education as basis, Stian and Jon are previously educated as electronic technicians.

1.5 Target groups

1.5.1 Application

The system has two usage applications, one as a centralized management solution, the other as a standalone configuration application. This results in two target groups for the software. The first group of users are network administrators who need to manage, configure and maintain firewalls based on OpenBSD and PF. The second group are SOHO[5] and home users, who need to configure a single firewall.

1.5.2 Report

The target group of the report consists of IT professionals, in the form of system and software engineers, network and system administrators. The primary purpose of the report is to provide a good documentation of what has been done and learned during the project period, and is in the interest of Gjøvik University College for evaluation of the end result, along with an external examiner.

The secondary purpose of the report is to document the requirements, the architecture, the design and implementation of the Copwall system. This is geared

[5] Small Office or Home Office

towards making further work on the system possible. A developer that who has not been involved in the process so far, should be able to use this documentation to get an understanding of the construction of the system.

1.6 Conditions

1.6.1 The project

This project is a graduate project at the studies for bachelor of engineering, computer sciences at Gjøvik University College, Department of Computer Science and Media Technology, who also is the formal contractor for the project. Contact person in relation to contractor is set to be Espen Torseth, while the project supervisor is Erik Hjelmås. The project period runs from January 3rd till May 19th, 2005.

The group has been allocated the room A036 at Gjøvik University College, and this will be the primary site for work activity. The room A036 will also be location for meetings, to the degree this is feasible. Work on the project will primarily be done the days Monday through Friday, core work time from 8am to 3pm, while this may be deviated according to general agreement within the group.

In terms of hardware allocated for the project, the group has been given exclusive access to eight computers of Pentium™ class, all with three or more network interfaces installed, and a total of two monitors and keyboards. These computers and «consoles» are the property of Gjøvik University College. In addition to this, group members use their private laptop computers. The need for three network switches will be met though the use of private equipment.

1.6.2 Distribution of responsibilities and roles

This is the distribution of responsibilities we have agreed upon:

- **Magne Andreassen** - Group leader and responsibility for web updates
- **Jon Langseth** - Responsibility for development, reporting and logging.
- **Stian Ramse** - Responsibility for documentation

1.6.3 Division of project tasks

The project development cycles will be divided based on the subsystems running on the client, server and the firewall. We use a top-down attack and start with the Client GUI. A large portion of the total time is given to this task. This includes GUI mock-ups, researching OpenBSD subsystems to be supported, prototyping, development and API specifications. Because of the size of the GMC subsystem, this is divided by two milestones to avoid long periods of time without a common group goal. The preliminary development task contains iterations for each main module we are to design and/or develop. Each of them will be reviewed at end

of the iteration, and all will be reviewed as a total at the end of the preliminary development period.

The CMP has an important function in the overall design, but the actual coding for this module will be postponed if needed. Same goes for the CAS subsystem. If the project suffers from time starvation, we will use the time reserved for development to document and design how it should work, rather than focusing on writing code. We prefer a good design without code, over poor design with buggy code.

On the 25th of April, we go into the closure period of the project. The main tasks in this iteration is finishing the graduate project report, and finishing the most critical software bugs and clean up code. These tasks will continue in parallel throughout the rest of the project period. Deadline for report and editing is set at Friday the 13th of May. After this day we focus on preparations for the presentation.

Apart from the milestones set by Gjøvik University College, the group has set its own milestones and decision stages. Please see the Gantt chart, included in appendix B, for further information.

1.7 About the employer

The idea for the project was first introduced to Einar Snekkenes, Professor of Information Security at Gjøvik University College. He was positive and forwarded the request to Erik Hjelmås. After introducing the project to Hjelmås, Espen Torseth was contacted and asked if he would act as the project employer, with Erik Hjelmås as teaching supervisor. Torseth accepted, and after a brainstorming session in November 2004, the basis for the project was set. Espen Torseth teaches the course Perimeter Security at Gjøvik University College, Master of Information Security, and has several years of experience from ErgoGroup[6].

1.8 Configuration management

1.8.1 Selected development standards, -model and -environment

The programming languages used in the project will be:

- **Java** for graphical user interface and end-user client application
- **C/C++** for middleware, management software and firewall administration software
- **Perl scripts** for system administrative tasks on firewalls
- **Bourne Shell script** for simple, elementary support functions

[6] http://www.ergo.no

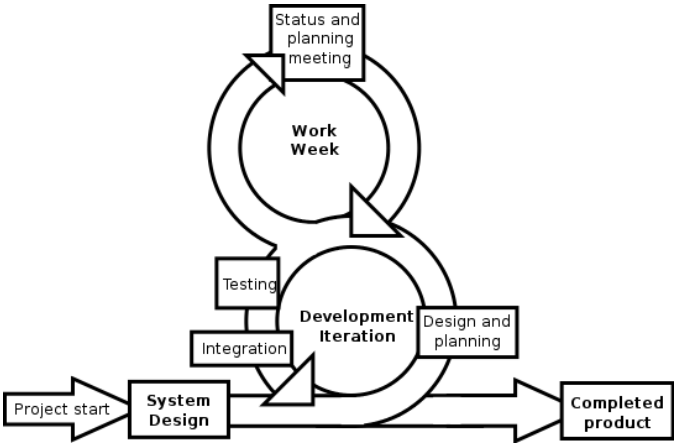

Figure 1.2: Illustration of the development model used.

The development model that is to be followed, is based on two variants of agile development. Extracting key elements from FDD [4] and Scrum [5], a model will be used that focuses on first defining the overall system architecture, design and feature sets, followed by incremental planning, development and documentation. Each increment is developed over one or more short loop iterations of five work days. Each short loop will be initiated by a summary, status and planning meeting, similar to that of Scrum. At the end of each increment, unit testing and integration is performed.

1.8.2 Structure of the report

The report is loosely constructed around the template published by Gjøvik University College, and contains a introduction, main section, closure, bibliography and attachments.

Chapter 1: Introduction An introduction to the project and the document in general.

Chapter 2: Requirement specification The formal requirement specification document of the system, along with a discussion of its contents and the work performed to produce it

Chapter 3: Design A description and discussion of the overall system architecture and design, and more detailed design discussion of each of the main modules.

Chapter 4: Implementation Describes the implementation work that has been done, and in some cases not been done, on the system.

Chapter 5: Testing Gives an overview of how system testing has been performed.

Chapter 6: Discussion of results In this chapter, the work and end result is evaluated against the requirements set for the system, and various problems or errors are discussed.

Chapter 7: Conclusion Concludes the project as a whole in context as a graduate project. Also provides a description of further work needed to complete the system.

Bibliography, definitions and Attachments

1.8.3 Standardization of documentation- and storage format and language

The documentation will be written in plain text using text-encoding ISO-8859-1, and will be formatted with LaTeX2$_\epsilon$, and filenames based on: theme-chapter.tex. The language we will be using is English, with little emphasis on Oxford or American sentence structure.

Version and revision control will be performed with use of CVS. This system handles all problems with editing the same document on the same time, and it gives the opportunity for comparing versions with inspection. CVS is used for both documentation and source code. Substantial themes or sub-projects will be separated in different areas within CVS.

Chapter 2

Requirement Specification

2.1 Requirement Specification document

2.1.1 Introduction

Copwall is a multi-layered software system for graphical configuration and administration of single firewalls and centralized configuration of multiple, distributed firewalls. The first version will be a system for managing OpenBSD based firewalls, with management features including the configuration of operating system, hardware and firewall related parameters. It is desirable that the system allows future expansion with support for platforms other than OpenBSD, and also allowing expansion with additional software subsystem support. This must be obtained by using a modular design, based on the concepts of plugin modules and software agents.

2.1.2 Function

The software is set to be built as a three tier architecture (see figure 2.1) on top of the OpenBSD operating system and other essential supporting services, such as database engine or virtual machine. The three tiers of the system will be:

- GMC, a graphical user interface that will typically be used on a non-OpenBSD workstation connected to a firewalled network.
- CMP, a middleware system that can be installed on a separate management server, or directly on a computer acting as firewall
- CAS, to be installed on computers performing firewalling, and handles the actual administration of the firewall computer and/or firewall cluster.

It will be possible to use the GMC as a standalone application, where all data generated in application specific format is stored locally, and data transfer to and

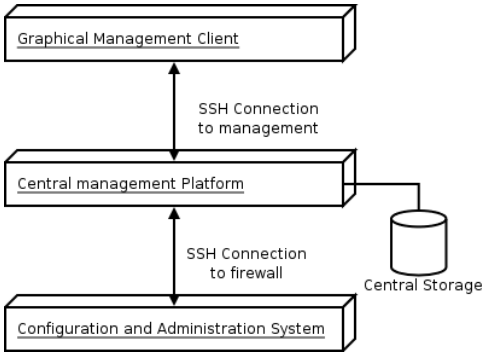

Figure 2.1: Main subsystems and communications. See text for details.

from managed firewalls are performed directly, without the intermediate management system.

When not using the software in a standalone configuration, all communication shall go via the intermediate CMP. The CMP is to be installed on a Unix based management server, preferably running OpenBSD. All generated data in application specific format is stored in a revision controlled repository managed by the CMP. This configuration is considered the standard scenario.

All transfer of data between the subsystems will be done over encrypted network communication, using SSH as the mechanism for encryption. As the arrows in figure 2.1 shows, all communication is bi-directional.

2.1.3 System environment

The GMC will be run by the user on a desktop or workstation computer, capable of running J2SE[1] 5.0, and supporting TCP/IP based networking.

The CMP will run on a computer running OpenBSD, and requires CVS, OpenSSH and libssh to be installed on this computer. Network communication may be performed via any TCP/IP capable communication channels with sufficient bandwidth.

The CAS will be installed on firewall computers. As such, the software will run under the OpenBSD operating system, requires the computer system to have a minimum of two network interfaces, and should be (but is not required to be) connected to an uninterruptible power supply. If the software system is used to configure a firewall cluster containing two or more physical computers in one logical firewall, each computer in the cluster is required to have a minimum of three network interfaces.

[1] Sun Java 2 Platform Standard Edition

2.1.4 The users of the system

The end users of the software are network administrators who need to manage, configure, manage and maintain firewalls based on OpenBSD and PF. These users are assumed to have a fair knowledge of computer usage in general, and in addition at least a basic knowledge of the purpose and function of a firewall.

2.1.5 Life cycle aspects

During development, CVS is used for version control and configuration management. From the CVS, regular developer or experimental releases should be made, and these releases must be tagged in CVS. Stable releases and stable patch revisions must also be tagged in CVS.

All parts of the Copwall system must be developed in a modular way, but it is not required to use dynamic pluggable modules (plug-ins). Added or removed modules that results in changed functionality are provided by changing version (release) of the software. As such, to gain access to new functionality, the user is required to upgrade to a new version of the software.

Version numbering should be used to denote differences in features and compatibility, and identify the claimed stability of a release. A version numbering scheme based upon the *major.minor.patch* scheme with odd numbered unstable releases will be used, similar to that previously demonstrated by the Linux kernel versioning scheme [6].

Major changes in features, architecture or GUI, and changes rendering files from previous versions of the software incompatible or unreadable must be denoted a major version number. Minor feature additions, where file format compatibility is kept, must be denoted a new minor revision number on the current major revision. Bug fixes are given a patch revision number. Code considered to be stable is given an even number in the minor revision number. Developer, unstable and experimental releases are given a revision with an odd minor revision number. When an odd numbered minor revision is considered stable, it will result in a new patch level release in the even numbered revision chain if no new features are added. If new features are added, a new even minor revision should be made, with a patch level of zero, and the next unstable or developer release should be given an odd minor revision number, counting one higher than the minor revision number of the latest stable release. Major changes causes a new major number, but only when this code is released in the stable version number chain. Releases prior to initial stable release should only have minor.patch version number, with minor set to zero.

As an example: Prior to initial stable release, all developer releases are given a version number of 0.x, where x starts with 1 and increments by 1 for each patch or release. The first stable release is given version number 1.0.0. Further development is done on versions 1.1.x. A stable patch revision would increment the stable release to 1.0.1, while continued increments are used on the unstable 1.1.x chain. Upon a new minor feature change, version 1.2.0 is released, and development re-

leases start with version numbers 1.3.x. Upon a major feature change release , or a compatibility changing release, the stable release moves to version 2.0.0, and developer releases to 2.1.x [2].

The version numbering of the various subsystems should be coherent. This means that a GMC of a given major.minor release must be able to be used with any release of CMP with the same major.minor, and the same goes for compatibility between CMP and CAS. Exception to this rule is accepted for the developer release chain (odd numbered minor number), but it is required that any developer release that requires a specific version of a different subsystem clearly note this in its changelog.

2.1.6 Constraints

The GMC requires a computer capable of running J2SE 5.0, with a minimum of 50MB available hard disk space for installation. It requires a screen resolution of 1024x768 pixels. A minimum of 256MB installed system RAM is required for acceptable operation. As the GMC is a network intensive application, a network connection with sufficient bandwidth is required.

The CMP system requirements are dictated by the requirements of underlying software. This software includes the OpenBSD 3.6 operating system with OpenSSH, libssh, CVS and PostgreSQL installed. Storage space requirements are dictated by the size of deployment, and as such impossible to predict. An assumption is made that 5MB storage minimum should be allocated each managed firewall, to allow revision handling of configurations sufficient space, plus 20MB for the CMP itself. If the CMP is installed on a computer not also acting as a firewall, this computer is required to have a minimum of one network adapter.

The CAS requires OpenBSD with OpenSSH, Perl, Python and libssh installed. The CAS requires a minimum of 5MB available hard disk space, and should not require more than 20MB of storage on the firewall. Each firewall that is administered by Copwall through CAS must have a minimum of two TCP/IP communications, e.g. two network adapters, or a network adapter and a WAN adapter. The networking adapters used, must be supported by OpenBSD.

2.1.7 Assumptions

It is assumed that computers used for developing, testing and running GMC are desktop class computers capable of running J2SE 5.0, and that J2SE 5.0 JRE[3] is installed. On systems used for CMP and CAS, OpenSSH and libssh is assumed to be available, and it is assumed that OpenBSD's syslog service may be used for logging.

[2] Version numbering is a debated topic, see http://www.advogato.org/article/40.html

[3] Java Runtime Environment

2.1.8 Detailed Requirements Specification

Functional structure and inter-relationships

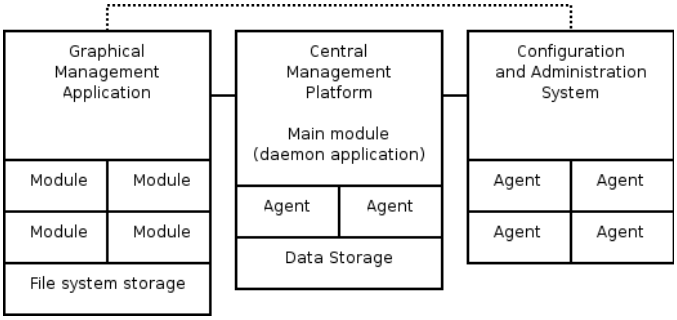

Figure 2.2: Subsystems modularity. Dashed line shows communication when CMP is not used.

Figure 2.2 shows the proposed system structure. The system must consist of three separate subsystems. These systems should in turn be divided into smaller sections, each handling a separate function or operation.

This division shall be done as software modules inside a single application for the GMC. On the CMP and CAS, the division may be selected more freely, but it is suggested that it should be based on a central application responsible for communication between subsystems, and «agents», separate applications controlled by the central application.

Data framework

Data used internally in the system shall use XML in all stored files. These XML files naturally must be parsed and translated into application internal data structures that are dictated in great extent by the programming language used.

Data exported and transferred to managed firewalls must be translated from XML format to the configuration format expected by the managed system. In example, a filter configuration for OpenBSD 3.6 is generated for storage in XML format by the GMC, stored in that format at the CMP, and translated from XML to the PF syntax format when transferred to the CAS on the firewall.

The format of the XML files should be controlled by XML Schema Definition (XSD) to ensure correctness, and Document Type Definition (DTD) files should be used to simplify translation from XML to system specific format.

At time of writing it has not been decided wether the GMC or CMP should perform config generation. This will be noted here in a future release of requirements, but developers should assume that GMC will receive this responsibility.

Graphical Management Client requirements

Usage description, input/output

The GMC is, as the name implies, a GUI application. Its intended primary role is as a front-end application, used to display and edit configurations stored and administrated by a central management server, the CMP. The application may also as previously described, be used as a standalone application, used to manage firewalls without the need for the central management server. The same GUI is used for both modes of operation. What follows is a user oriented description of how the application should operate, and how it should be used.

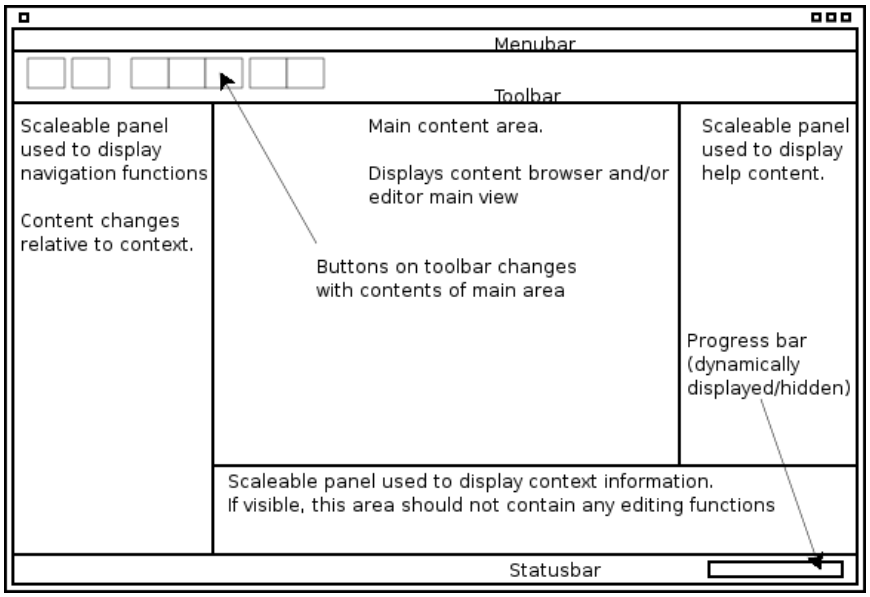

Figure 2.3: Mockup of main window.

Figure 2.3 shows the elements of the core GUI elements. The application will consist of the following elements:

- A toolbar
  Buttons representing available operations will be added to and removed from the toolbar by the different modules that make out the GMC application. The buttons should always represent navigational operations, and selection insensitive operations. No operations that manipulate a single, selected item on screen should be present on the toolbar, as these operations should be provided in a pop-up context menu, accessed using a right click on the item to be manipulated.

- A menubar
  The menubar will provide access to all or most available editing, manipulation and navigation operations. The length of a single menu on the bar should

not exceed 16 menu items, including sub menu items and separators. The use of sub menus should be kept at a minimum, and a sub menu should not contain more than 8 items, including separators. Nested sub menus should be avoided completely. The number of menus on the bar should not exceed 16.

- A statusbar
  As the familiar name of this element implies, this bar should display the current status of the application. On the right hand side of the bar, a progress indicator will be displayed on time-consuming operations. For operations that use a long time to complete, but have a duration that is impossible to predict, a continuous operation progressbar is displayed.

- Four resizable areas
  Naming these four areas *west*, *center*, *east* and *south*, the following content will be applied:

  - Center:
    The main area of the application, and the natural focus point for the user. This is where the main functionality of the application is placed. I.e. all central editor display functionality for creating and editing configurations is placed here, and this is where all firewall groups and units are visually represented.
  - West:
    As the secondary area of the application, this area is used to directly manipulate the type of information displayed in the *center* area. Here alternate navigational aid is placed, e.g. a tree view. This area is also used to display buttons that manipulate what type of information is displayed when the main area displays a configuration editor.
  - East:
    Instead of using a separate window for viewing application help, the right most area is used to display assistive information. The behaviour of the help viewer is not yet completely decided upon, but it is suggested that this should dynamically change to reflect the contents of the *center* area, and the operation the user is performing.
  - South:
    The southern area is used to display console and log information when configurations are transferred to a firewall, This may be done in a real-time fashion, or displayed after transfer and activation is completed. This area may also be used to display extended information about a selected entity in a navigational view. Usage of this area should be kept at a minimum, as it steals valuable window real estate from the *center* area.

  All of these four areas support scrolling using GUI scrollbars. All areas except *center* may be automatically displayed or hidden through software,

depending on what areas are needed in a given context. The help view can be opened and closed manually by the user.

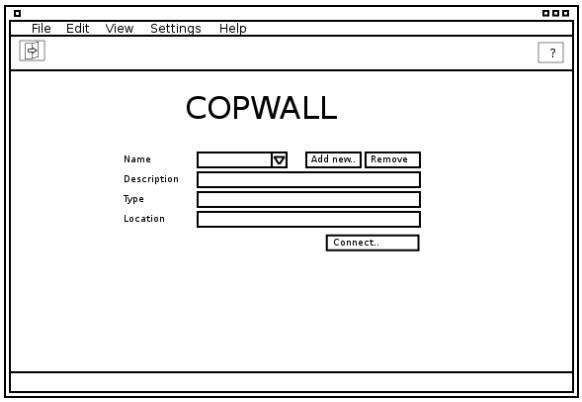

Figure 2.4: Mockup of connect view

When the application starts up, the main window is displayed with only menubar, toolbar, statusbar and *center* area visible. In the *center* area a connect view is displayed (see fig. 2.4). In this view the user can select a previously configured configuration repository or management server from a drop down list, optionally selecting a button to create a new location, or delete the selected location. After selecting the appropriate option, the user clicks a button labeled «Connect».

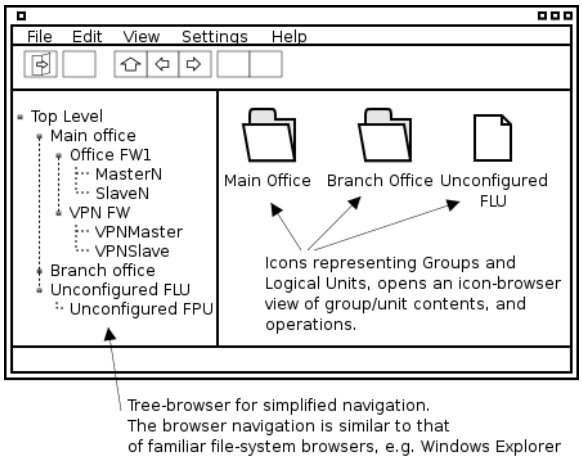

Figure 2.5: Mockup of navigation browser, viewing top level groups

Communication with a management server is then initiated, or the local repository is located and checked. A file containing info about managed firewall groups and units is loaded, and the navigation browser is displayed (see figures 2.5, 2.6 and 2.7). This browser view follows the familiar navigational style of a file manager, like Windows Explorer or Konqueror. In the *west* area, a tree view representing

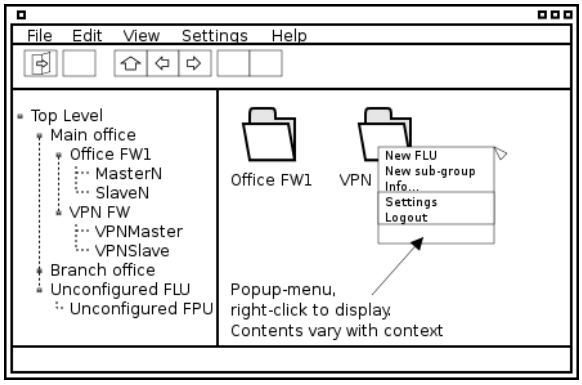

Figure 2.6: Mockup of navigation browser, subgroup view.

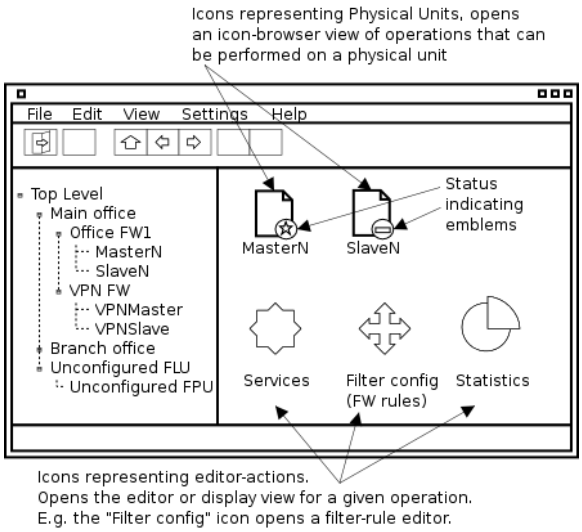

Figure 2.7: Mockup of navigation browser, showing contents of a FLU.

FLGs, FLUs and FPUs are displayed. FLG's may be nested, and may contain both FLG's and FLU's. Only an FLU may contain a FPU. A FLU cannot contain a FLG.

In the *center* view, medium sized icons representing the contents of the current group or unit is displayed. The navigator is operated using single mouse clicks. Right clicking on an icon brings up a menu that, in addition to the items available when right clicking in an empty space, contains options to manipulate the single item. In the navigator display mode, the toolbar displays buttons that allow the user to navigate backwards and forwards between displayed items. It also displays a button that allows navigation upwards in the item hierarchy. In addition to the navigational buttons, the toolbar contains buttons to add new items of type FLG and FLU. When displaying the contents of a FLU, both icons for FPU's and icons representing configuration editor actions are displayed in the *center* area, and on

the toolbar buttons to add physical units to the logical group.

When displaying the contents of a FLU, the icons representing FPU's are overlaid with visual emblems that indicate the current status of that computer. The icons representing configurations or views, are also activated using a single click method. These activate the display or editor module for the given type of information.

Within the view of a FLU, icons representing configuration of running services, editing of firewall filter rules, and display of system statistics and logs are available. Within the view of a FPU, icons representing system configuration (OS/platform), network interfaces, and logs are available. The operation of the configuration or editor views is as follows:

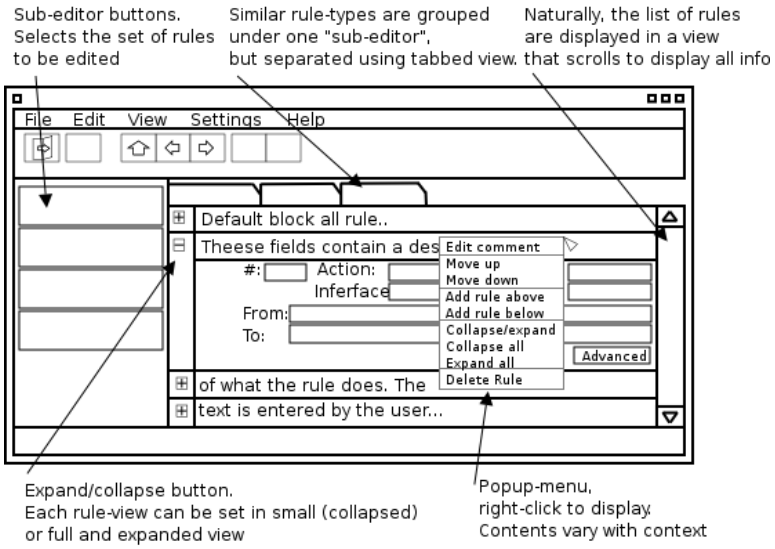

Figure 2.8: PF ruleset editor

PF rule editor (fig 2.8) The rule editor aims to move away from the typical table paradigm for firewall filter rule editing, providing an interface where central and common settings are directly and easily available, while maintaining availability of advanced settings.

Instead of using a typical table based view of filter rules, a vertically oriented list of graphical views is used. Each rule has a separate view, consisting of a label that the user defines to describe a rule, and a collection of buttons, entry fields and lists that together describe the central elements of a filtering rule. Each rule display may be collapsed, so that only the descriptive title, and perhaps a rule syntax summary of the rule is shown. Advanced rule editing is provided through a modal dialog box, that displays a full feature single rule editor. Ruleset editing functions, like adding, removing, and reorganizing rules, are provided using a context menu, available via right clicking on a rule.

Different types of filtering rules are separated into categories, and navigating between the categories is done using navigation buttons located in the *west* area. This navigation concept has previously been demonstrated in similar fashion by the Opera web browser by Opera Software[4]. The categories group conceptually similar rules, e.g. NAT, BiNAT and redirect rules under the same navigation button. Within a category, syntactically or functionally different filtering rules are separated using a tab based layout, e.g. placing NAT, BiNAT and redirect rules in three separate tabs.

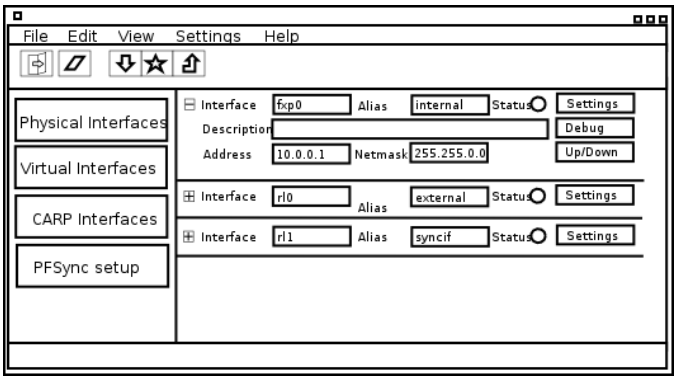

Figure 2.9: Network interface configurator

Network interface configurator (fig 2.9) The interface editor consists of navigational buttons in the *west* area, and a list of interfaces in the *center* area. The navigation buttons are used to select between types of interfaces displayed, where type is e.g. physical interfaces, virtual interfaces or CARP interfaces. In addition, navigation buttons should be available to provide direct access to the configuration parameters of CARP failover, VPN and VLAN configuration.

The list of interfaces consist of visual editors that are collapsible, with only the most central information visible in the list. Refined or advanced settings are still available through a «Settings ...» button in the visual display, that brings up a detailed dialog box. Each interface has its current up/down/error status indicated in the list, and in the expanded view, buttons are provided to allow toggling of enable- and debug-state.

[4]http://www.opera.com/

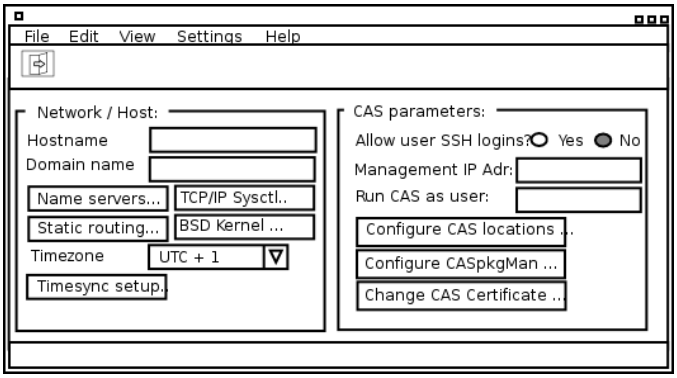

Figure 2.10: Platform and operating system configurator

Platform and operating system configurator (fig 2.10) The configuration editor for platform and OS specific settings closes away the *west* and *south* areas (user may keep help view open), and uses the remaining area as a large dialog box. In view is displayed grouped sets of configuration parameters that may be set. Types of platform settings that have a complex nature, should be edited in separate dialogs, and a button to open the dialog should be placed on the *center* area.

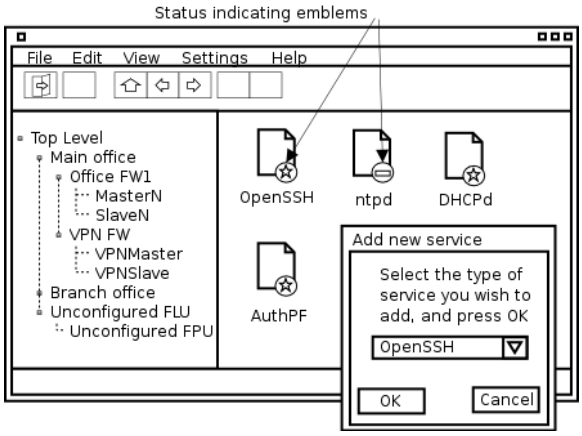

Figure 2.11: Available services configurations

Available services configurations (fig 2.11) When activating the Services view, an icon browser view is displayed, containing icons representing installed and configured services on a given FPU. Selecting an icon brings up a modal dialog box that is specific for configuring the given service. Adding a new service to a FPU is performed using a button on the toolbar, or selecting the appropriate option from a menu available by right clicking anywhere in the *center* area. Right clicking on an icon brings up a menu that allows the

user to start or stop, remove (deactivate and unconfigure), configure or add a service.

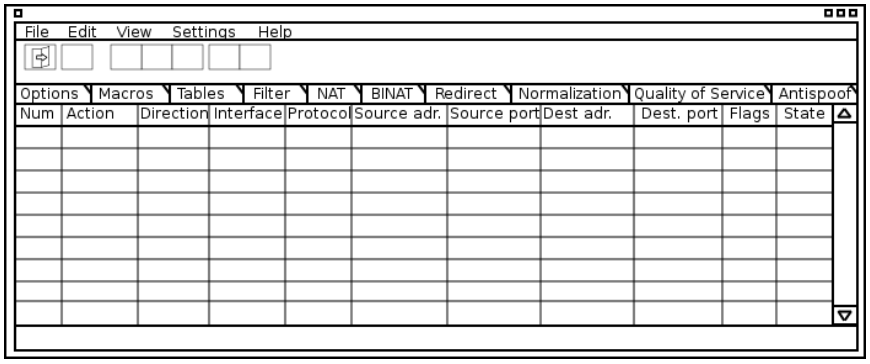

Figure 2.12: Tables based filter rule editor

Tables based filter rule editor (fig 2.12) Though it is not a requirement for the development of the first version of the system, the developers may choose to provide a standard tables based rule editor, and include this as an optional interface for the user. If this is included, it should use tab separated information pages, as shown in figure 2.12.

Performance, availability and error/failure reporting

The GMC is started manually by the user. It should never terminate unless requested to do so by the user. Exception from this is in the case of a severe software malfunction, where the application is allowed to perform an automatic termination, after first informing the user about the event, and what may have caused it through an informative dialog box.

In the event of a failure, the application must attempt to recover automatically. In the event of an unrecoverable error, the user must be notified, and the application should attempt to regain a stable state without termination. The overall stability will be to some extent be governed by the operating system the application is running on, and by the Java Virtual Machine used. Stability problems caused by OS or Java VM cannot easily be predicted.

Central Management Platform requirements

Performance, availability and error/failure reporting

The CMP will be centered around a continuously running application (a daemon). This daemon should have a very high level of availability. In the event of software failure in the daemon itself, or one of its subsystems failure, the CMP should automatically attempt to recover and regain a stable state without termination. If termination of the daemon and its subsystems is required, these systems should automatically restart. All inconsistencies in operation, failures or stability

problems must be logged. Relatively severe problems must also be reported to all currently connected GMC instances.

Configuration and Administration System requirements

Performance, availability and error/failure reporting

The CMP initiates communication with CAS on managed firewalls. This should start execution of the CAS management communication subsystem. This subsystem should never terminate before the CMP closes the connection. The rest of the CAS should be run on a scheduled basis, with fairly short and automated execution runs. Errors encountered during a scheduled run must be logged.

Constraints details

All software libraries used in software through dynamic or static linking is required to have a licensing policy that does not put licensing constraints on the finished product. The same applies to all software systems that the Copwall system communicates and interfaces with.

Acceptable licensing policies are e.g. the BSD family of licenses, the Lesser GNU Public License, the Common Public License, the Eclipse Public License and the Sun Public License. Licenses that enforces openness or closedness on the end product should be avoided, e.g. libraries released under the GNU Public License shall not be used.

Applications used to develop code and/or documentation is required to be released under a license that is approved by the Open Source Initiative. Exception to this is the operating system of the computers used as developer workstations, though it is required that a valid license is provided for the operating system on each workstation.

The following programming languages are suggested:

- J2SE
- C++
- Perl
- Python

The following storage formats and transaction languages are to be used:

- XML
- SQL
- plaintext configuration formats, e.g. pf.conf

The following external systems may be used:

- OpenSSH
- CVS
- PostgreSQL

The following libraries are suggested:

- JSCh
- Castor
- libssh

2.2 Description and discussion of Requirement Specification

The requirement specification document is set as a combined product and requirements description, and is focused on a usage view. This means that the operation and function description is written as a description of how the finished product is expected to function. This way, the requirement specification fulfills its task of highlighting the expected functionality and requirements, and in addition functions as an introduction to the system usage, and as a marketing document aimed at recruiting developers to the project.

The formal requirement specification document did not exist until quite late in the project period. The contents of the specification has existed throughout the project span, and has been under continuous revision. But it has been a collection of informal notes and sketches, along with unwritten understanding between the group members. With such a small group, we have not seen the need for a formal document before late in the project. We see now that since the system itself is not completed, it is necessary to formalize the information into a requirements document that other developers may use, to continue further development. As the current group will be somewhat split up after the graduate project period is over, and some time may pass before further work is done, the document is also necessary for ourselves to keep a written record of the agreed requirements and functionality.

As the need has not been seen for a formal requirement specification document earlier in the project period, the requirement specification document has become a rather large bulk of work, concentrated on a short time span. This is naturally not ideal, and we are fully aware of this.

Chapter 3

Design

3.1 Introduction

While the project was still in its planning phase, before being accepted as a graduate project assignment by Gjøvik University College, the system was planned as a far simpler solution. The original idea was to create an application for creating filtering rulesets for PF for single OpenBSD based firewalls. At the early stage, support for multiple firewalls was not included in the plan, it focused solely on single machines. When starting discussions with our assigned contact person, the idea of a centralized management solution, supporting the configuration and administration of multiple firewalls was introduced early. This drastically changed the size of the project, but also lay down the architecture of the solution quite early.

This chapter describes and discusses the architecture and design of the Copwall system. As the goal of the project was to create a sound foundation for a complete system, the language used is to great extent set in a future tense.

3.2 The overall architecture

The system architecture is a three tier system. This does not mean that it employs the classical three tier architecture of application front-end, middleware system and back-end server. In Copwall, the three levels are application front-end, management server and middleware, and managed client systems. The front-end application is the GMC, which is the application that the user interacts with to manipulate information stored at the management server, and to instruct the management server to apply changes onto the managed firewalls.

It is possible to exclude the management server, and use the GMC with direct communication with managed firewalls. When a management server is not used, all configuration data generated by the GMC needs to be stored locally on the computer running the application. Figure 3.1 shows the architectural setup where a management server is used.

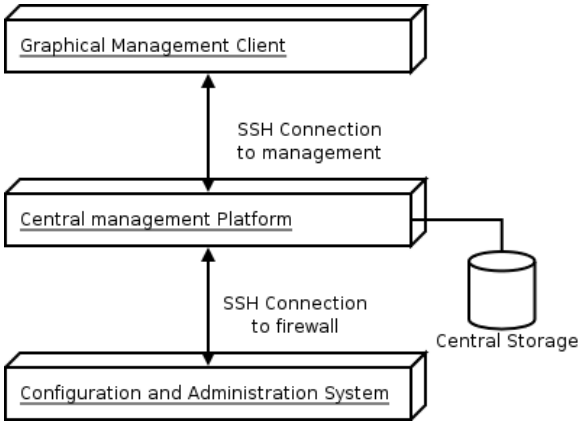

Figure 3.1: Main subsystems and communications.

All communication in the system needs to allow bi-directional data transfer. Commands issued, and responses to command, along with status information should use the same channel. Communication between an instance of the GMC are to be continuous, to allow quick responses to user actions, and to allow the transfer of management initiated status messages. Communication between a managed firewall, and the management system does not need to be continuous, and should not be, to preserve bandwidth. With the logic that the management server should transfer configurations and commands to the firewall, and the firewall should transfer logs and status information to the management server in bulks, there should be no need for a continuous communication.

3.3 The choice of communication encryption

The system has a high requirement of secure communications. This is natural taking into consideration that what is being configured and managed, is in fact the first line of defence of a network. Using communication that is not encrypted and authenticated in some way, will quite simply allow someone with malicious intents to read out and possible change configurations and statuses, and in that way making the firewall nothing more than a porthole into the network behind it.

Several methods have been discussed for encryption and authenticating communications. Initially the plan was to use secure socket layers (SSL) to implement this. SSL is a security protocol that provides communications privacy over the Internet. The protocol allows client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery [7].

Use of SSL in C/C++ code for simple encryption of transferred data was simple and rather painless, and though there was some more work involved in getting it to work with Java, it seemed a very viable solution. But when we tried getting the SSL connection authenticated, witch is done using X.509 certificates, things started

getting complicated. We wanted bi-directonal authentication, but could only get authentication of certificates to work uni-directional. When we combined this with problems we saw in getting bi-directional data transfer over a single communication socket, we started evaluating other mechanisms.

One such mechanism that had been suggested early on was revisited. The SSH protocol suite provides secure communications that have been thoroughly tested, and proven to be a relatively safe choice. With our basis in OpenBSD, the OpenSSH implementation of SSH2 is readily available. After a short research we located two additional resources that eased our adoption of the SSH2 mechanism for encrypted communications. libssh is a library for inclusion of SSH2 communications in C and C++ based applications. It has a simple API, and is released under the Lesser GNU Public License. For Java applications we found the JSCh library, that allows advanced SSH2 communications to be easily integrated. JSCh is licensed using a modified BSD license. One great advantage we found in using SSH as transport mechanism, is that a single communication session may be used to transfer several channels of data. For example, a single session may simultaneously transfer application commands and responses, in parallel with file transfer in a separate channel. Even multiple command interactive channels may be open simultaneously on a single session. SSH2 may be configured to use a combination of host and user public key authentication, where RSA key pairs are used. This leads to a form of bi-directional authentication, where both server and client public keys are confirmed, after encryption has started, but before data transfer initiates.

Combining the problems we saw in using SSL, with the overly complex task of researching and developing our own solution for encryption, and the advantages that SSH2 communication gave us, the choice of SSH as transport mechanism was a simple one. The result is that all network based communication channels in the design are assumed to be SSH2.

This choice in turn, does dictate some other design issues. OpenSSH will be the server daemon that controls incoming SSH connections, and will in turn start up applications that the connecting client request. Thus all software that is to be controlled via SSH, needs to be developed in such a way that it expects its input/output to be handled as it was controlled on a local terminal. As a result, software on the CMP and the controlling software on the firewalls (part of the CAS) is programmed with input/output via STDIN/STDOUT/STDERR. File transfers will be handled via either SFTP or SCP channels.

This also gives an advantage in terms of testing and debugging, as all elements of commands that normally would go encrypted over a network, may be tested locally on the system it is to run, as if it were a standard, console operated application.

3.4 Central Management Platform

The purpose and function of the CMP is to be an intermediate system that support storage, revision control of configuration files and data, status monitoring and log file storage and is an intermediate task broker.

A *job* or *job request* consists of a task description, the commands needed to complete the task, configuration files and information about when and how to execute.

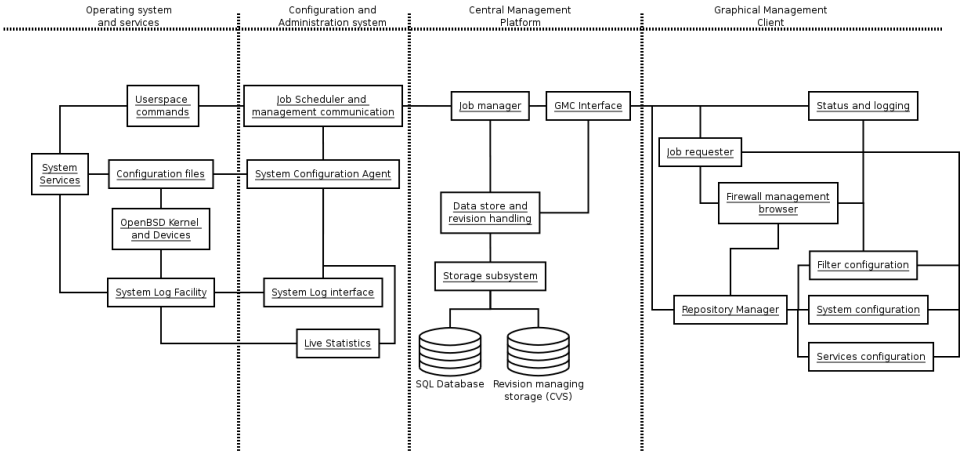

Figure 3.2: Main system design overview.

The construction of the Copwall CMP is split into separate applications and software agents. The three central elements in the CMP are the GMC Interface (abbreviated GMCI), that management client applications interact with, the Job Manager (abbreviated JM), that executes actual management of connected firewalls, and the Data Store and Revision Handling system (abbreviated DSH), which is a middleware system that handles extraction and storage of data in the underlying storage systems.

3.4.1 Data Store and revision Handling system

The use of a middleware DSH enables masking of the underlying data storage system, and thereby makes the other parts of the CMP independent of storage mechanism. In the event that storage mechanisms are replaced, the only part that needs to be rewritten is the DSH, in stead of having to rewrite all parts of the CMP that uses the data store.

3.4.2 Graphical Management Client Interface

The GMCI is a program that is started upon connection from a GMC. The actual startup of the GMCI is handled by OpenSSH2, as a result of a SSH subsystem

request from the connecting client. The GMCI remains running while the client application is connected. The job of the GMCI is to receive commands from the GMC, and respond appropriately to the command. This means to :

- receive configuration files and other data from client, and pass this off to DSH for storage
- retrieve such data from store through the DSH, and transfer to GMC
- receive job requests and transfer these requests to the Job Manager (JM) subsystem.
- transceive job execution feedback from JM to connected GMC

3.4.3 Job Manager

The JM is a continuously running part of the CMP system. Its responsibilities is to acquire information about tasks that are to be performed on the managed firewalls, and transfer information about these tasks along with configuration data across an outgoing SSH2 connection to the firewalls. The JM can schedule jobs independently of the GMC, and this is why the JM is a continuous running system. Implementing the JM as a continuously running system also eliminates the need to use systems like cron to perform scheduled, non-interactive operations within the system. An example of a job that is regularly executed without an explicit request from a GMC, is the collection of log files from the managed firewalls. When a GMC is connected to the CMP, a job that is requested directly from the GMC is expected to be executed immediately, and in a singe SSH connection. The GMCI uses Unix pipe pairs to communicate with the JM, and will normally keep this communication open while a client is connected, to allow data transfer from the JM through the GMCI to the connected client. Configuration files are not to be transferred through the GMCI/JM pipes. The GMCI will transfer all other parts of the job description through the pipes, and the JM will use this information to extract the correct configuration files and relevant data through the DSH.

When the JM transfers a job to a firewall system, it will start an SSH2 connection to the firewall, and activate the Job Scheduler (abbreviated JS) as an SSH subsystem. The SSH connection will remain open after the job request is transferred, and is used to transfer job output back from the JS to the JM. This output may be textual output from executing commands, textual streaming of data from the firewall's syslog, or a file transfer in the form of log file or configuration file. When the job is requested from a connected GMC, output in form of text stream is transferred to the GMC via the GMCI and the Unix pipes. When the job is a «non-interactive» one, the output stream is buffered into a log of the job run, and stored into the data store. Both for interactive and non-interacive jobs, file transfers in the form of log files or configuration files are stored via the DSH.

3.5 Configuration and Administration System

The CAS is split into smaller components, just like the CAS is. The central component is the Job Scheduler, which handles all communication with the CMP, and is responsible for starting execution of the other subsystems. The System Configuration Agents (abbreviated SCA) are the actual executing parts of the CAS. These are specialised pieces of software that perform the operations needed by a job.

An executing session of the CAS is initiated by a SSH2 connection from the JM on the CMP. The JS is then started as an SSH subsystem, and job specification along with configuration files are transferred. The JS remains running, keeping the connection with the CMP open until all tasks in the job specification is either completed, or confirmed failed.

3.5.1 Job Scheduler

When the JS receives a job specification, it waits until an identifier telling it that the specification is fully transferred before starting execution. It then executes one or more of the system configuration agents, in required order, to get the job done. Output from the SCA's are transferred back to the CMP's JM using the open SSH2 stream, prepending each output line with the string `SCA:` followed by a space. The JS also reads all lines that appear in the system logs during its run, and transfer these to the CMP, prepending each line with the string `LOG:` followed by a space. Status output from the JS is prepended with the string `OUT:` and a space. In the event of an error, this is notified to the CMP using a message with the string `ERROR:` prepended, and during development and testing, debugging messages may be sent with the string `DEBUG:` prepended, both followed by a space after the colon. Whenever a file is requested to be transferred, the JS will not perform the actual transfer, it will make a copy of the file, using a generated, unique file name, storing it in the JS temporary storage area, then inform the CMP that the file is ready for transfer by sending the string `FILE:` *`filename filespec`*, where filename is the name used for the file in the temporary storage area, and filespec is a specification of which file this originally was, e.g. *pf.conf*. For log file bulk transfer files, the *filespec* part should always be `logfiles`.

During the transfer of a job specification from CMP to CAS, the JS will, as noted not start execution of the job until it is fully transferred. It will, however, return status information, informing the CMP line by line whether the line is accepted or rejected. These status messages are constructed by the keyword `STATUS:` followed by a space, a status indicating number, a space, and last a text string representation of the string. The following messages may be expected:

- STATUS: 100 Connection accepted
- STATUS: 101 Welcome to Copwall CAS
- STATUS: 101 CAS version x.x.x

- STATUS: 102 JS version x.x.x
- STATUS: 200 OK
- STATUS: 210 Job received OK
- STATUS: 215 Job execution START
- STATUS: 220 Job execution COMPLETE
- STATUS: 400 Invalid command
- STATUS: 410 Invalid keyword
- STATUS: 415 Wrong number of parameters
- STATUS: 420 Wrong parameter format
- STATUS: 500 Server failure
- STATUS: 501 Not ready
- STATUS: 502 SCA not available
- STATUS: 555 SEVERE FAILURE

These status numbers are grouped by the de-facto standard grouping used in most plain text communication protocols accepted by the IETF Network Working Group.

- **100** Informational messages
- **200** Success and acceptance messages
- **300** Redirection messages
- **400** Error in request from client
- **500** Server errors

The JS runs with normal user privileges, i.e. no root or super-user privileges. When the JS is to execute an agent that requires root or similar higher privileges, the agent is executed through a wrapper program that utilizes the suid and sgid system calls to change the effective user-id that the agent executes at. This wrapper script is a compiled C or C++ program, and has compiled in constants that contain a checksum and byte-size for each of the agents that may be executed through the wrapper. This ensures that the wrapper under no circumstances will execute an agent that has been tampered with. The developers must pay notice to this, if an agent is modified, the checksum and byte-size constants of the agent must be updated, and the wrapper must then be recompiled.

3.5.2 System Configuration Agents

The SCAs are primarily written using the Perl scripting language. This allows for rapid development, and easy scripting of advanced functions. Perl was among the first scripting languages to gain acceptance in the Unix world, and has, to some extent, replaced much shell programming as the Free Software lingua franca of system administration [8]. Perl combines features of the C programming language, with a number of extra features that make it ideal for dealing with text files and databases. This makes Perl an essential tool for system administration tasks [8]. The standard availability of Perl on the OpenBSD platform makes it a natural choice over alternative scripting languages like Python.

3.6 The job specification language

The structure of this job specification format is: first a controlling keyword, second the primary parameter, and on some keywords further additional parameters. The definition of the job specification syntax is attached in appendix F.

A sample job-specification is as follows (line numbers not part of spec):

```
 1   job begins
 2   job id 3933350
 3   file pf-841587000 pf.conf
 4   file ssh-072572000 sshd.conf
 5   backup pf.conf
 6   backup sshd.conf
 7   install pf.conf
 8   install sshd.conf
 9   rehash
10   wait 5*60
11   rollback pf.conf
12   rollback sshd.conf
13   rehash
14   log genbulk
15   log transfer
16   job ends
```

This is a basic job specification used to make a «test-run» of two configurations. In this job run, the CMP transfers two files to a the firewall, placing the files in the temporary storage space, and naming the files `pf-841587000` and `ssh-072572000`. On lines 3 and 4, these files are specified to be used as configuration files for PF and SSHd, respectively. On lines 5 and 6 the system is instructed to make backup copies of the currently installed configurations, before installing the new files on line 7 and 8. Line 9 tells the system to reload all managed configurations, thereby applying the new configurations. Next a 5 minute wait period is specified. After the wait, the last backup copies of the `pf.conf` and `sshd.conf` configurations are put back in operation, and applied on line 13. Line 14 and 15 generates a log file bulk package, and prepare these files for transfer. Line 16 ends the job spec. Only when this command line is received from the JM, the execution is started.

3.7 Graphical Management Client

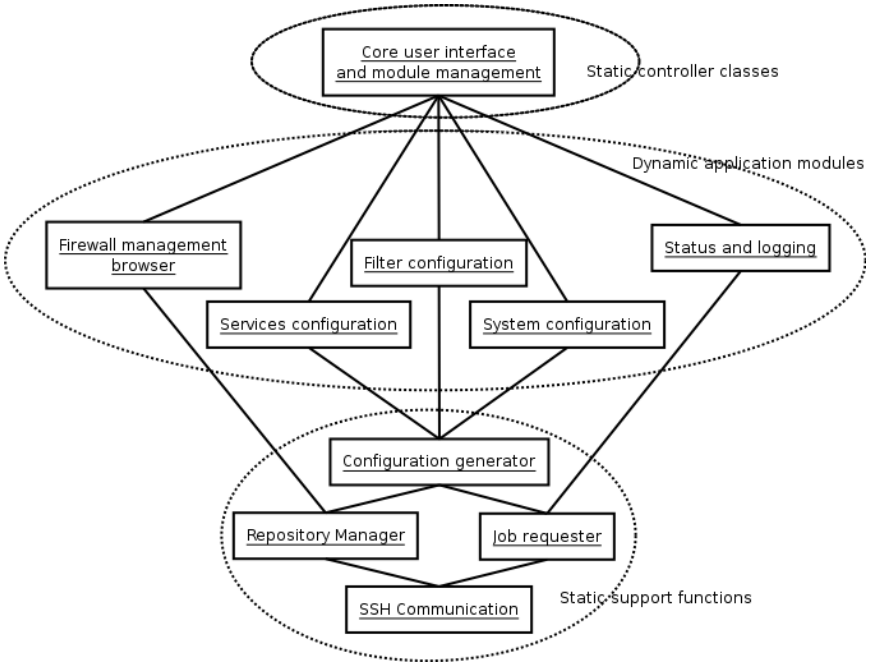

Figure 3.3: Architecture of the GMC system.

The GMC construction consists of a collection of core functionality, a set of modules that extend the core interface and functionality, and finally a set of support functions and modules. The core functionality includes elements like the main GUI framework, startup functionality and module management. The central element of the core is the core user interface, which is instantiated at application startup. Most of the elements in the core are implemented as static classes. This means that functions and variables located in the core, are globally available in the application, and data is persistent during the execution.

This construction is also used for the supporting modules. As an example, one of the supporting functions is a module for SSH communication. This module needs to use a single SSH connection for all communication, while multiple of the other parts of the application need read/write access to this connection. We find that this is most easily implemented by using a static construction in Java, as opposed to building a message and object passing architecture. The supporting functions are not considered to be part of the core, as they do not need to be present at all times, and as such are instantiated on first use. Interfacing between the supporting functions and the other modules also go directly, and not through the core classes.

Figure 3.3 shows the architectural division of modules, and where communication between modules occur. Separate modules are created for each type of information to be displayed on screen.

3.7.1 The design of module loading/selection

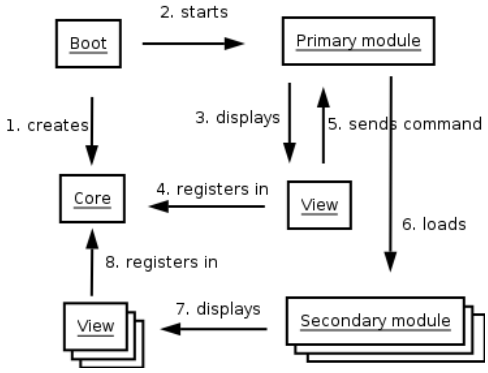

Figure 3.4: Flow chart of boot process and module loading.

What the user sees as the application, is actually the user interfaces of the modules, loaded onto the core user interface. Except for the window framework and visual divisions, the core and support functions should be transparent to the user. All application interaction is done through one or more application modules. One such is selected as the primary module, and is started at application startup. This primary module allows the user to select how the application will be used; as a standalone filter editor tool, or a front-end for a management solution. The primary module is next responsible for loading either the firewall management browser module, or loading the filter configuration editor. The application startup and module loading is illustrated in figure 3.4.

When the application is up and running, selection of displayed module is managed by the core module management. Selecting a new type of operation, makes the currently active module to request the loading of the appropriate module. The requesting module is then kept active, but is hidden from view. When the new module is loaded, it registers itself with the module management. Once the user is done using a module, this module is removed from view, and unregistered from the module management. The core user interface will then bring into view the module that was responsible for opening or activating the just closed module.

3.7.2 Data structures in the Graphical Management Client

Application logic dependent data structures are not restricted by any general design guidelines, except from those imposed by the libraries and classes used. There does however exist a clearer design of the data model related to representing firewall groups and units, and the information contained within these. Figure 3.5 shows how the data model is conceptually built up. A tree structure is used, with a data object representing the management host at the top, and groups and units following.

As a firewall may be configured in a failover solution, the differentiation of logical units and physical units is done. As such, a logical unit is what totals as a

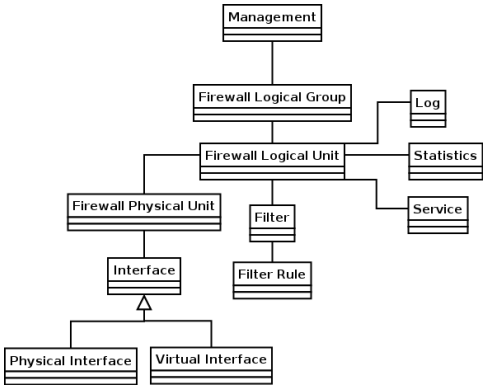

Figure 3.5: Domain Model view of the Graphical Management Client, simplified representation.

firewall solution, while a physical unit is a single host, acting as a part of a logical unit. Information that is common for all hosts involved in a logical unit, is kept and presented to editors at the logical unit level, while information specific to a host is presented at the physical unit level.

It must be noted that the data model pictured in figure 3.5 is the architectural model. In implementation there is no need for an actual data binding between the nodes in the structure. As an example, the firewall browser module only uses a limited part of the data tree, as shown in figure 3.6. The browser module is however the location from which the filter editor, log viewer and service configuration modules are started. Thus the browser module is the starting point for loading these parts of the data.

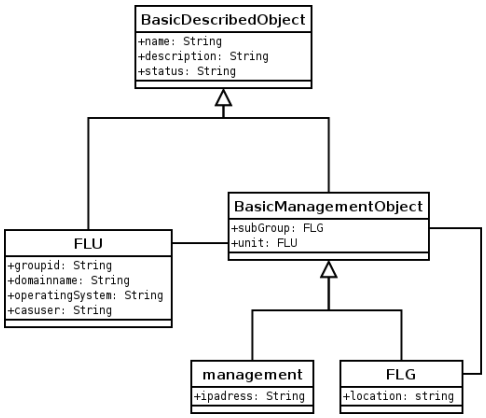

Figure 3.6: Class diagram of the data model for representing firewall groups and units administrated using a Copwall system.

Please note that figure 3.5 is a simplified domain view of the data model. The actual data class models are far more complex. A separate class diagram is created

for each of the significant data elements, and an XML Schema Definition document is set up to implement the corresponding data model, both for Java code, and for the XML files used for persistent storage of data. A sample of such a class diagram and corresponding XSD can be found in the appendixes D and E.

3.7.3 Application code package separation

The application code is split into separate Java packages, following a logic of separating different modules into separate packages (fig. 3.8). The purpose of this is to impose a logical structure of where various source files may be located, and to enforce an awareness of scope and code separation onto the developers. As Java package separation in itself is not a rigid code separation tool, the developers are directly responsible for making sure that low coupling[9] is achieved, by defining and documenting interfaces between packages. At this stage in the design process, there has not been created any formal interface specifications.

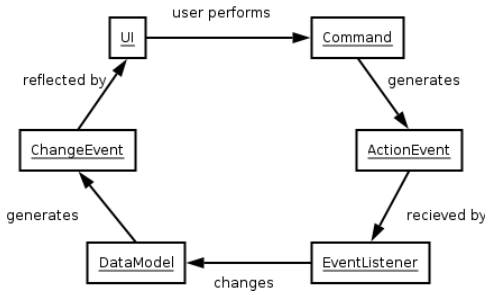

Figure 3.7: Detailed model-view-controller data flow.

Within each module package, there is further package division. As a model / view / controller construction is used in the application (fig. 3.7), there should be package distinction of data model, user interface, and controlling code within each module. Therefore each module package has at least three sub packages, «ui» containing user interface (view) components, «actions» containing the controlling command classes, and «model» containing the relevant data model classes. Variations to this theme is allowed. For instance does no data model package exist in the core package, as this has no central data model. The core package does however contain additional packages like «widgets» and «commands», containing elements that are not specific to any particular module, but used generally in the application.

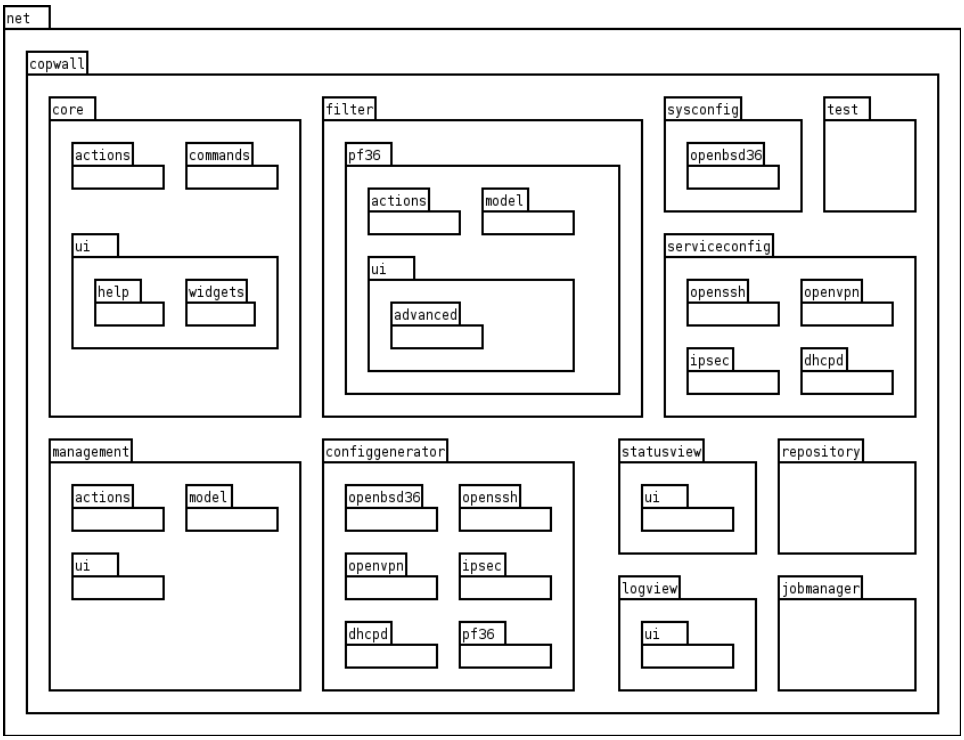

Figure 3.8: Package structure of Java code in Graphical Management Client.

Chapter 4

Implementation

4.1 Development tools and IDE used

For the source code in Java, we have standardized on the Eclipse Platform by the Eclipse project. Eclipse is an extensible platform for integrated software development, developed primarily for Java development, but supporting a wide range of other languages and features through a plug-in system.

Specifically our configuration of the Eclipse Platform uses the Eclipse Software Development Kit (SDK), version 3.0.1, including Eclipse Platform runtime, Plug-in Development Environment (PDE), Java Development Tools (JDT), Rich Client Platform (RCP) and Standard Widget Toolkit (SWT), extended with the C/C++ Development Tool (CDT) the Eclipse Modelling Framework (EMF) and support plugins for UML modelling support and LaTeX syntax support.

4.2 Third-party Java libraries used in the project

The following third party libraries have been used in development of Java code, and must be bundled with the finished application to make it fully functional:

- Castor version 0.9.6
  `http://castor.org/`
  XML parser and generator with support functions.
- Castor depends on the following libraries from the Jakarta Project
  - Commons Collections
    `http://jakarta.apache.org/commons/collections/`
  - Commons Logging
    `http://jakarta.apache.org/commons/logging/`
  - ORO
    `http://jakarta.apache.org/oro/index.html`

  - Regexp version 1.1
    `http://jakarta.apache.org/regexp/index.html`
  - Xerces-J version 1.4.0
    `http://xml.apache.org/xerces-j/`

- Windows Look and Feel Fidelity 0.5.1
  `https://winlaf.dev.java.net/`
  Look and feel bug fixes for Swing on the Windows platform.

These libraries are either licensed under the BSD License[1], or under one of the Apache license versions 1.0, 1.1 or 2.0[2]. In addition to the previously mentioned libraries, the Java Secure Channel, JSCh library will be used for SSH2 communication in Java code.

During the start of the development we evaluated a few other libraries and packages, that we thought could simplify the development. One of these was SwiXml, a small GUI generating engine for Java applications and applets. With SwiXml, the GUI is described in XML documents that are parsed at runtime and rendered into javax.swing objects [10]. While this tool early on seemed to give us a tool allowing for rapid prototyping of user interfaces, it was soon concluded that implementing rather complex GUIs using SwiXml provided a relatively small gain, compared to the effort needed to learn the structure and use of the XML files needed to SwiXml. Thus we have only performed a short investigation and test of SwiXml, and not used it in further development.

To allow for scalability of the GMC, we intended to use a plug-in based architecture, supporting dynamic extension of the application. As our knowledge of dynamic class loading and Java re-entrant code was rather limited when the project started up, research was made trying to find an existing framework for a dynamic plug-in architecture. What we came up with, was the plug-in framework of Eclipse, and the Java Plugin Framework (JPF). The Eclipse Plugin Development Environment (Eclipse PDE) is a powerful, but large system that allows for dynamic applications. While trying to comprehend the API specification of this library, we realized that using this tool would make the resulting solution heavily dependent on most of the Eclipse platform. We found that not to be acceptable for our application. A longer examination and test was therefore done on the Java Plugin Framework. This is a framework that allows for simple integration of plug-in functionality in a Java application. But using this as a central part of the application, would make internal data transfer within the application rather cumbersome. Combining this with the lack of time we had available to learn how to use it efficiently, this was also scrapped.

As noted, the third party libraries used in the Java application, will have to be bundled with the application upon distribution. It is preferable that the distribution of the Java application be done as a single JAR file. Due to limitations in the

[1] http://www.opensource.org/licenses/bsd-license.html
[2] http://www.apache.org/foundation/licence-FAQ.html

Jar format, it is not possible to directly package additional Jar files within one Jar package. To accomplish this, an additional tool is needed. The de-facto standard solution to this problem, is to use One-JAR[3] package created by P. Simon Tuffs. We will use an Eclipse plug-in that automates the use of One-JAR, called FJEP, the Fat Jar Eclipse Plugin[4] by Ferenc Hechler and others.

4.3 Code conventions

4.3.1 Start comments

All java source files should start with a C-style comment containing the CVS `Id` tag, and creation date:

```
/*
 * Id
 * Created on ${date}
 *
 */
```

The `Id` tag is expanded by CVS on first commit, and should look something like:

```
Id: CoreUI.java,v 1.48 2005/05/10 13:40:52 magne Exp
```

The `${date}` tag is a Eclipse internal variable and is expanded on creation.

4.3.2 Naming conventions

We have aimed at following Sun's naming conventions explained in the *Code Conventions for the Java™ Programming Language* [11] document. This document reflects the Java language coding standards presented in the *Java Language Specification* [12], from Sun Microsystems, Inc.

4.3.3 JavaDoc

For Source Code comments, standard JavaDoc is used. All classes, interfaces, constructors, methods and significant variables should have a short JavaDoc comment to explain its purpose.

For any other in-line comment needs, `//` single line or `/* comment */` multi line C-style comment may be used where applicable.

[3] http://one-jar.sourceforge.net/
[4] http://fjep.sourceforge.net/

4.3.4 Source code example

```
/*
 * Id: MyClass.java,v 1.1 2005/05/10 13:40:52 magne Exp
 * Created on 15.feb.2005
 */

package net.copwall.mystuff;

import net.copwall.yourstuff.YourClass;

/**
 * Class description goes here.
 *
 * @version xxx
 * @author Name
 */
public class MyClass extends YourClass {

    /** firstVariable documentation comment */
    public static Object firstVariable;

    /** secondVariable documentation comment */
    protected SomeClass secondVariable;

    /**
     * MyClass constructor comment...
     */
    public MyClass() {
        // code here...
    }

    /**
     * doSomething method comment...
     */
    public void doSomething() {
        // code here...
    }

    /**
     * doSomethingElse method comment...
     * @param myParam description
     */
    public void doSomethingElse( Object myParam ) {
        // code here...
    }
}
```

4.3.5 Class imports

Importing external classes should never be used implicit like e.g. `import net.copwall.yourstuff.*;`, but explicit like this: `import net.copwall.yourstuff.YourClass;`. This can make the import block rather large in some circumstances, but makes reading and understanding what external classes are used for rather easy. Eclipse supports auto importing of classes, so organizing imports is an easy task for the developer.

4.4 Core functionality

Core functionality is collected in the `net.copwall.core` java package. This package contains the core functionality used throughout the application.

4.4.1 Boot sequence

The Boot sequence is a rather simple implementation responsible for setting up logging, initiating core components like CoreUI, displaying a simple splash-screen while loading the different modules like the ruleset editor, management and repository. We have stressed the usage of the logger in Boot, and as a result all debugging and log messages are correctly logged to file during operation. Under no circumstances `System.out.println` should be used, except from temporary, restricted debugging. The Boot process will first try to start CoreUI, and thereafter all registered modules. If everything loads correctly, CoreUI is set visible to the user. If an exception is thrown, the error message is logged to file for further investigation.

4.4.2 Core User Interface

CoreUI is the main window for Copwall, and is implemented as a singleton instance. It can be accessed in a static, synchronized way to keep it thread-safe. By making the modules responsible for adding themselves to the different views, like e.g. *center* area or *west* area, CoreUI is only used as a repository for the current visible modules. This concept should eventually support storing the different views so that switching between them without reloading is possible. Different public methods is implements to add components to the views, and these methods also manages resizing and splitpane logic. Splitpanes are visual dividers separating the different views and supports resizing of the views. CoreUI also holds the instances of the main menu, toolbar and statusbar. Members in need of adding components to any of these, must fetch them trough CoreUI. Some logic is added not to break the order of menu item groups and toolbar placeholders, but the developer sometimes has to override these if placing elements in non-supported areas.

4.4.3 User action events

Collecting action commands in a sub-package called `.actions`, developers are encourage to place all user supported actions here to clearly show what actions are initiated by the user, and what actions are generated by members.

4.4.4 Custom widgets

PopDownButton

A PopDownButton is basically two buttons operating as one. As figure 4.1 illustrates, clicking the right part of the PopDownButton, a popup-menu is displayed underneath it. Clicking the left part of the button will in this particular situation bring up the advanced settings for the rule action.

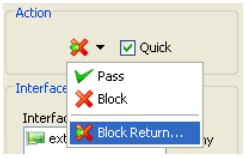

Figure 4.1: Action popup menu

XPanel

XPanel is a JPanel extension capable of operating in expanded or collapsed mode. In collapsed mode, only the top component is visible, and by clicking the plus-icon, the panel expands and reveals the bottom component. The same action can also be accomplished trough double clicking the top component. See figure 4.2 for the conceptual layout of XPanel. The collapsed and the expanded components can be any Java Swing `Component` of choice. XPanel is the core component for filter rules.

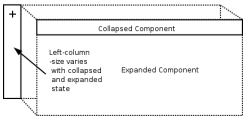

Figure 4.2: XPanel layout

SettingsDialog

The abstract SettingsDialog is created to keep setting dialogs consistent trough the application. Should a settings dialog be needed, the developer must subclass this class. The abstract methods `okPressed()`, `applyPressed()`, and possibly the `cancelPressed()` must be implemented. SettingsDialog defaults to being a modal dialog. The developer can override this at request.

4.4.5 Command framework based on the command pattern

The package `net.copwall.core.command` contains the command framework (fig. 4.3) used for coupling user inputs with actions that should be executed. The command framework, is an extension of Swing's own action framework, to not break compatibility with Swing components. To meet the requirement of a complex user interface with hundreds of different input actions, a simple way to associate a button or a menu item with text and possibly an icon was needed.

An abstract command holds information about the command to be executed on a user input, as well as the text label to display to the user and optionally a icon, tool tip text, mnemonic key and an accelerator key. This means that both a button and a menu item can be associated with the exact same command. Actually the command can be associated with any Swing `AbstractButton` implementing components like e.g. a `JButton`, `JCheckBox`, `JToggleButton` and so on.

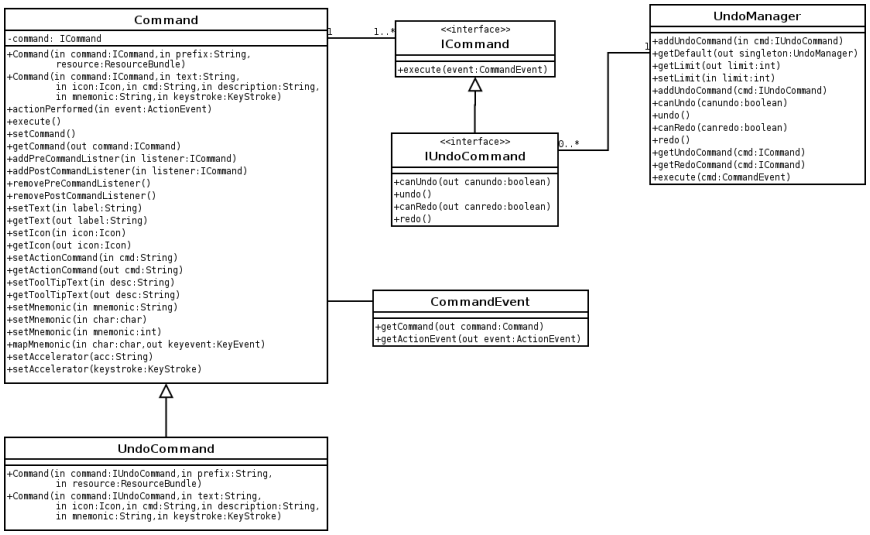

Figure 4.3: Command Class Diagram

Internationalization of commands

The reason for extending Swing's `AbstractAction` was to internationalize the text and keyboard accelerator keys easy. Information about the command can be stored in a `ResourceBundle`, and following a given pattern, completely define how the command is displayed to the user. By using `ResourceBundles` to define command faces, they are 100% internationalized, and text, icon, mnemonic and accelerator keys can easily be changed without modifying source code.

`ResourceBundle` file: `messages.properties`

```
file.open.label = &Open
file.open.icon = net/copwall/core/ui/icons/16/open.png
file.open.desc = Open Packetfilter
file.open.acckey = ctrl O
```

`Open` is used as the label for the command, `open.png` as the icon, and so on. The `&` precending `Open` tells the Command to automatically create a mnemonic identifier for the character `O`.

Following code will create a new command based on the data given in the `messages.properties` file:

```
ResourceBundle res = ResourceBundle.getBundle( "messages" );
ICommand runMe = new ICommand() {
    public void execute( CommandEvent event ) {
        System.out.println("runMe was executed!");
    }
};
Command myCommand = new Command( runMe, "file.open", res );
```

When creating a new button based on `myCommand` we only need one line of code, and the same command can be reused in e.g. a menu (figures 4.4 and 4.5):

```
JButton myButton = new JButton( myCommand );
JMenuItem myMenuItem = new JMenuItem( myCommand );
```

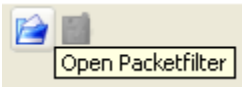

Figure 4.4: A JButton created with the abstract Action «myCommand»

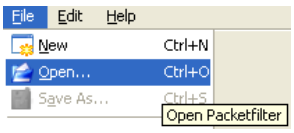

Figure 4.5: A JMenuItem created with the abstract Action «myCommand»

Processing of user input

The response to an user input event, will trigger the `actionPerformed()` method in Swing, and eventually execute the command associated with the menu and/or button. To be more specific, pressing any of the *myButton* or *myMenuItem* will trigger the `actionPerformed()` method in Swing, which in turn will execute the `runMe` object and print `runMe was executed!`.

In its simplest form, the `ICommand` interface implements one single method; `execute( CommandEvent e )`. which is executed from a Swing `actionPerformed` event. For an undoable command, we need to implement some additional methods, which we will look at soon.

Pre / post command listeners

To be notified before or after a command is executed, a pre or post listener needs to be registered with the Command. The pre and post listeners implement the exact same interface; `ICommand`. See figure 4.6. The pre and post listeners are passed a copy of the `CommandEvent` object to be notified what command is about to be executed, or was just executed, respectively.

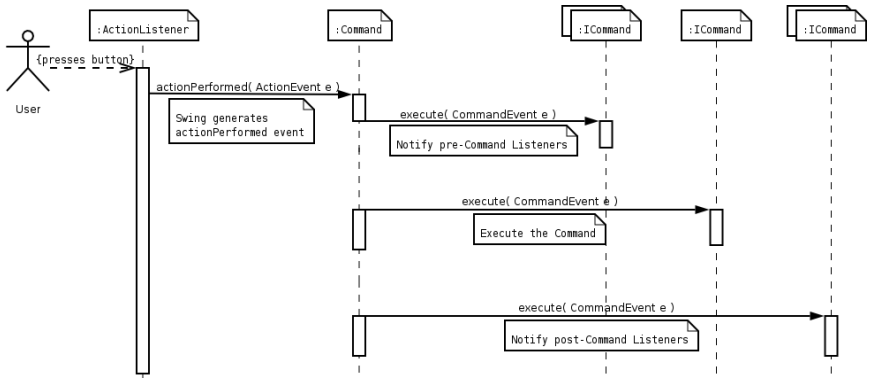

Figure 4.6: Pre- Post CommandListener Interaction Diagram

Undoable commands

Some user input events needs to be undoable, and by extending the command framework and adding the `UndoManager`, this is accomplished by listening for execution of a command, and put it in the undo queue. The `UndoManager` is simply a post command listener (fig. 4.6), listening for undoable commands.

If an `UndoCommand` is executed, the `UndoManager` registers the command in its undo queue and waits for the user to undo it. Initially, the queue holds 100 objects, which should be sufficient in most situations. The developer writing commands, must implement the `IUndoCommand` interface and is responsible

for storing pre and post states of the model so that an undo or redo action works as expected.

4.5 Java and XML data binding

4.5.1 Packet Filter model

XML data binding is the shortest path between java objects and XML documents. The concept is explained thorough in the article «XML and Java technologies: Data binding with Castor» [13].

The PF model uses Castor and data binding. While meeting the requirements for a XML structured middle-format, we also gain advantage trough only storing away the data interesting for the ruleset. Using Castor's Source Code Generator, the PF model is auto-generated from the XSD Schema as seen in appendix E. Auto generating source code has some advantages and some drawbacks. The first test, we ended up with about 200 auto-generated classes for the PF model. Later, we discovered Castor's class inheritance feature, and we agreed to refactor the XML schema, and generate a new model. We accomplished to reduce the number of classes to about 90, and also gained a much better design.

We configured the source code generator to auto-generate `PropertyChange` methods for all classes. If any of the values in the model is changed, an event is issued to all `PropertyChange` listeners. This is an important feature for the MVC concept used in the application, and a more in-depth explanation is presented in the following subsection.

The PF BNF [14] was used as reference when creating the XML schema for Castor. Also the PF User Guide [15] was used frequently.

4.5.2 View / controller components

All GUI components viewing and controlling the model in some way must comply to a number of rules.

First of all the view / controller (VC) must be able to be initialized as a dummy VC, not connected to the model in any way. This should place the VC in a disabled, non-editable state, but laying out the various components if possible. From here on, when the model is added to the VC, it initializes the creation of all objects depending on the model, reads the current state, and register itself as a `PropertyChangeListener`. Controller components of interest, should be set to an enabled, editable state. See figure 4.7. The VC is responsible for correctly displaying the initial state of the model, and thereafter wait for change events or user input. We refer to this as the enabled state.

Controller elements like buttons, checkboxes and menu items modifying the model in any way, should never change their state based on other controller components, but rather listen for `ChangeEvents` from the model. In some cases

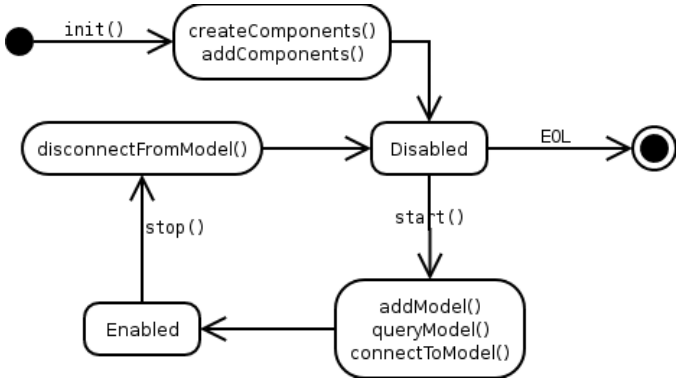

Figure 4.7: The GUI view / controller lifetime.

this is not possible or appropriate e.g. where one wishes to prevent user interaction during a running process. If the component supports drag and drop, it is crucially important that they implement the VC interface. This interface assure that the method `disconnectFromModel()` is executed before the transferable Object is serialized. If the VC component is not disconnected from the model, the serialization process will try to serialize the GUI components registered as `PropertyChangeListeners` as well [5]. As a general rule it is desirable that all VC components implement this interface to assure it can connect and disconnect from the model at request.

Delegated view / controller example excerpt

```
/**
 * Constructor sets ActionMenuUI in a
 * disabled un-editable state.
 */
public ActionMenuUI() {
    super();
    init();
}

/**
 * Constructor takes a rule as parameter.
 * Try to start the GUI component.
 * @param filterRule the PFFilterRule to view.
 */
public ActionMenuUI( PFFilterRule filterRule ) {
    super();
    init();
    setFilterRule( filterRule );
}

/**
 * Set the PFFilterRule to view.
 * @param filterRule PFFilterRule object
 */
```

[5] http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=4202657

```
public void setFilterRule( PFFilterRule filterRule ) {
    if( this.filterRule == filterRule ) {
        stop();
        connectToModel(); //reconnect to the model...
    } else if( null != this.filterRule ) {
        stop(); //implies disconnectFromModel();
        this.filterRule = filterRule;
        load(); //reloads and connects to model...
    } else {
        this.filterRule = filterRule;
        load();
    }
}
```

4.6 PF ruleset editor

The ruleset editor as shown in figure 4.8 needs some additional explanation. The figure is marked with numbers ranging from 1 to 6 to distinguish the individual components explained.

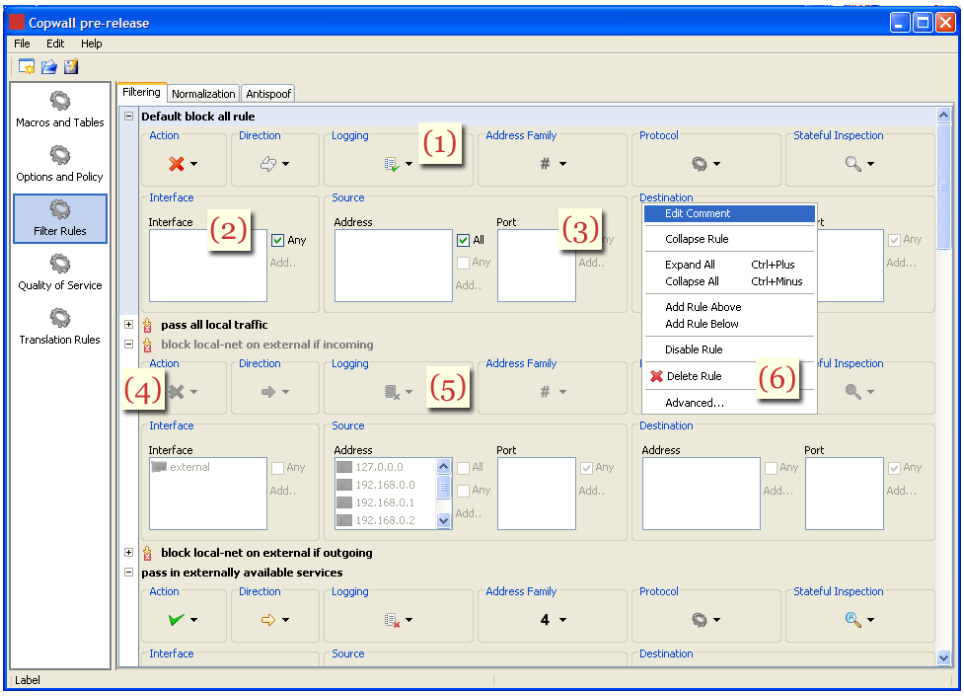

Figure 4.8: Screenshot of Copwall, PF ruleset editor.

4.6.1 Expanded rules

A single rule logically separates the different settings trough use of titled borders. Each of them explained below.

Action

First, the *Default block all rule* (Figure 4.8 `(1)`) states that this is a block rule, both trough the comment and trough the *Action* icon. From the «pop-down» menu displayed when clicking the down arrow right of the icon, the user is presented with options for selecting either «Pass», «Block» or «Block Return...» which in turn will open advanced action settings.

Direction

The *Direction* icon is grey, and represents that this rule will match any direction. The other options selected from the «pop-down» menu are «Inbound» and «Outbound» traffic.

Logging

The *Logging* icon is represented with a small green check mark in the lower right corner, indicating that this rule should be logged. The other options include «No logging» and «Log All». «Log All» is only applicable for stateful rules, and disabled for stateless rules.

Address Family

Address Family is either IPv4, IPv6 or any of them, here represented with a # indicating any.

Protocol

The *Protocol* section is where the user specifies which protocols this rule should match, again a grey icon represents any protocol. The preliminary supported protocols are «TCP», «UDP» and «ICMP». More should be supported.

Stateful Inspection

Stateful Inspection is also represented with a grey icon, indicating that no state inspection is used for this rule. The options are; «Keep», «Modulate» or «Synproxy State». A more in-depth explanation is discussed later in this chapter.

Interface

The *Interface* (Figure 4.8 `(2)`) section displays a list of interfaces this rule should apply to, possibly none if the *All* checkbox is checked. All available interfaces can apply to the rule.

Source and Destination

Source and *Destination* (Figure 4.8 `(3)`) sections are basically the same. Only thing separating them, is the *All* checkbox placed right of the Source Address list. *All* tells PF to totally ignore source and destination addresses, equivalent to any source and any destination. The Address list can consist of address(es), hostname(s), table(s) or interface(s). Any of these may be negated to invert the sense of the match, trough a popup menu. The Port list may hold single numeric ports, a port name (as specified in OpenBSD's `/etc/services` file) or a port range.

Ranges can be specified as exceptions, matching every port except the range specified, or as ranges including or excluding boundaries.

Comment

The comment is displayed as a un-editable label, but can be changed trough the *Edit Comment* menu-item in the popup menu. In edit mode, the label is swapped with a text field as shown in figure 4.9, to allow the user to edit the comment. The text field is swapped back with the label when the user is finished editing, defined as; `[enter]` pressed, or focus lost on the text field.

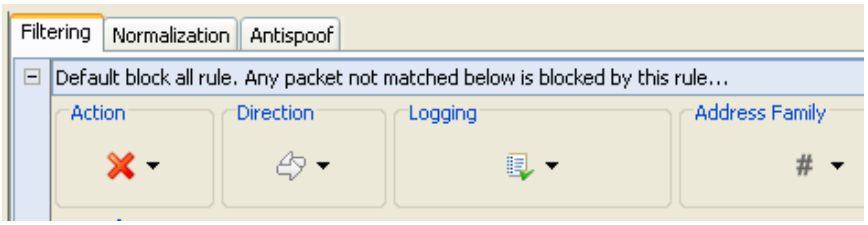

Figure 4.9: Editing the comment for a rule.

Left column

The left column (Figure 4.8 `(4)`) in the Ruleset editor is used to mark the current active rule, as well as holding the plus / minus icon for expanding and collapsing the rule.

4.6.2 Collapsed rules

The second rule *pass all local traffic* is collapsed, only showing the comment. It also has an icon arrow located in front of the comment. This icon represents a *quick* rule, in other words, this rule is evaluated as the last matching rule. Last matching rules (or quick rules as they are called in PF) are recently heavily debated on the PF mailing-lists [6].

4.6.3 Disabled rules

When disabling a rule, (Figure 4.8 `(5)`) it is suggested that the rule no longer be part of the «active» ruleset and is therefore left out, or commented out in the `pf.conf` output file. (The XML to `pf.conf` translator is as of today, not yet implemented.) Disabled rules are greyed out, and controller elements like buttons, checkboxes, etc are disabled to clearly state the fact that it is not in use.

[6] http://www.benzedrine.cx/pf/index.html

4.6.4 Dragging and dropping a rule

The rules can be rearranged trough drag and drop. When a drag gesture is recognized, the rule GUI VC components are decoupled from the model preparing for transfer, thus the rule look disabled to the user. A transparent border is painted over drop targets as the user moves the rule down or upwards the ruleset. When dropping a rule on top of another, the rule is added or moved to the index preceding the drop target.

4.6.5 Rule popup menu

The popup menu (Figure 4.8 `(6)`) supports general settings for the rule and rule set. The menu items include, editing the comment, collapsing and expanding the rules, adding, disabling, deleting and opening advanced settings for rules. The popup menu is built as general as possible, and should support all types of PF rules.

4.6.6 Advanced Rule Settings

Action

As figure 4.10 shows, some additional settings for the Action statement exist. For a block rule, the user may choose to drop the packet, or return an ICMP message to the source host. With TCP connections a TCP Reset message can be issued to terminate the connection. The *Default Block Policy* option will use the default block policy defined for this ruleset.

Stateful Inspection

The following text is a summary of the original OpenBSD `pf.conf` manual from the subsection `Stateful Inspection`. For a more in-depth explanation of Stateful Inspection, please see the original document [14].

Stateful Inspection is a method for tracking the state of a connection. This is used to eliminate the need to evaluate a complete ruleset, by inspecting only the initial package of a connection, and thereafter keeping state of all packages for this connection. All traffic for the connection will flow trough the filter because of the filter's awareness of it. By checking the sequence number of the packets before any rules are evaluated, the packets are passed or dropped based on the expected values. This prevents spoofing attacks and takes load of the filtering engine since no additional rules need to be evaluated.

Keep, modulate and synproxy state all share the same type of options, but operate in a different nature. Keep state, can be used on both TCP and UDP connection, though UDP is stateless by nature, the filter engine matches state based on source address and port. Modulate state is used for TCP connections where initial sequence numbers (ISNs) are assumed to be poorly generated, and vulnerable to

exploits. PF creates «a high quality random sequence number for each connection endpoint» [14]. Synproxy state is used for TCP connections where the user wants to make the handshake transparent to the active and passive endpoint. The advantage gained with synproxy state is summarized in the previously mentioned document as:

> No packets are sent to the passive endpoint before the active endpoint has completed the handshake, hence so-called SYN floods with spoofed source addresses will not reach the passive endpoint, as the sender can not complete the handshake [14].

As seen in figure 4.11, it is possible to limit concurrent state for the rule. The *State Binding* specifies whether the state should be bound to a single interface, a group of interfaces or «floating» between interfaces.

Figure 4.12 show the source tracking options for the stateful rule. Source tracking makes it possible to track state entries on per. source host basis, and specify limits if applicable.

Figure 4.13 show the TCP connection settings for a rule. TCP connection settings can e.g. limit the creation of new connections over a given interval originating from the same host.

Figure 4.14: Overload policy makes it possible to place a host, violation any of the limits specified, in a table to block further activity from the offending host or redirecting it to a tarpit process or restricting its bandwidth.

Figure 4.15: Flushing of states terminates the connections from the hosts violating any of the limits specified. The «Flush States Globally» option implies killing all connections from the offending host.

TCP Flags

TCP flags are most commonly used for matching packets initiating a new connection, but could also be used to match other flags set in the TCP header.

The following TCP flags are supported by PF:

- FIN - Finish; end of session
- SYN - Synchronize; indicates request to start session
- RST - Reset; drop a connection
- PUSH - Push; packet is sent immediately
- ACK - Acknowledgement
- URG - Urgent
- ECE - Explicit Congestion Notification Echo
- CWR - Congestion Window Reduced

By specifying TCP flags for check/mask (Figure 4.16), the rule can be matched only to packets having «check» set out of those specified in «mask». The «SYN / SYN, ACK» combination is often used to match a packet attempting to start a new connection. Looking at packets with only the «SYN» and «ACK» flags set, only «SYN» may be set. I.e. the combination would match a packet with «SYN» and «URG», but not «SYN» and «ACK». A mask must always be specified and GUI logic helps to accomplish this.

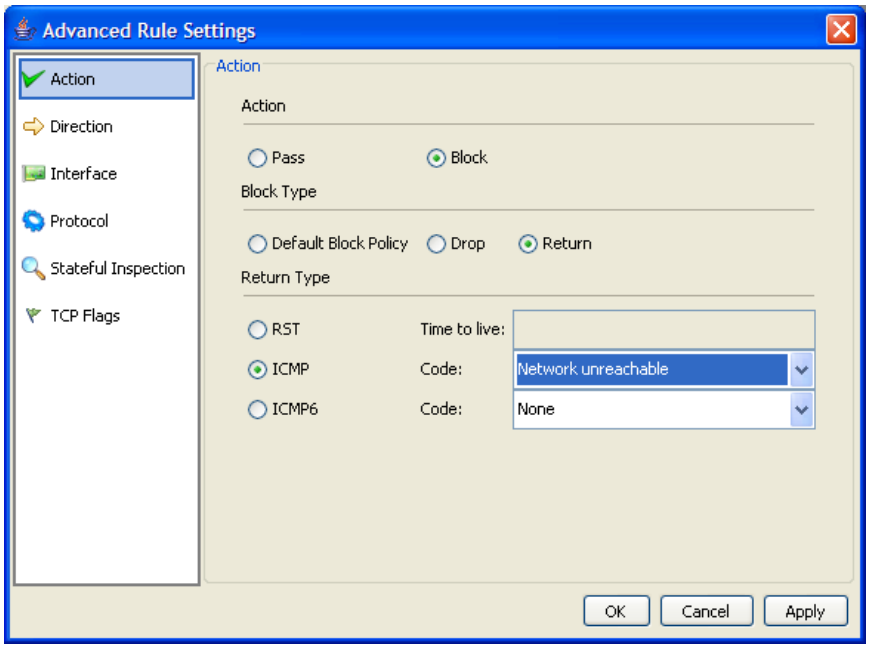

Figure 4.10: Advanced settings for rule action.

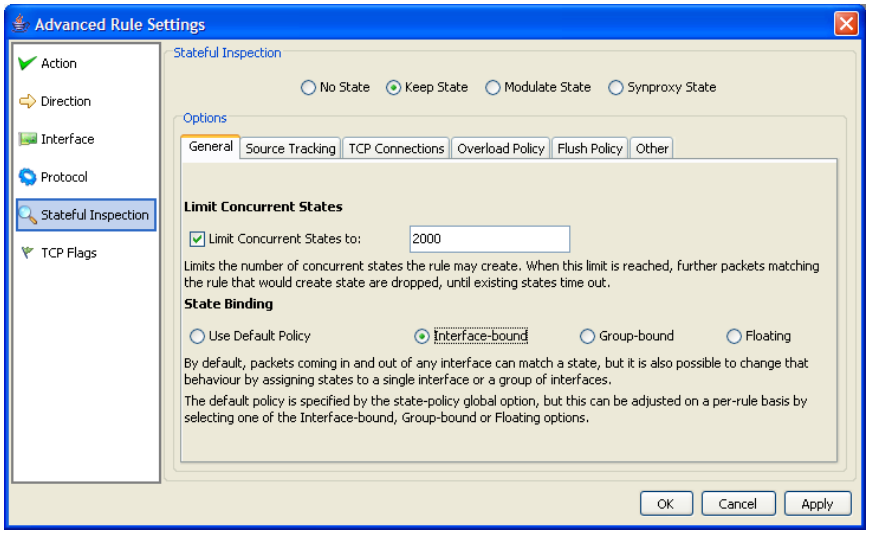

Figure 4.11: Keep State and limit concurrent states to 2000.

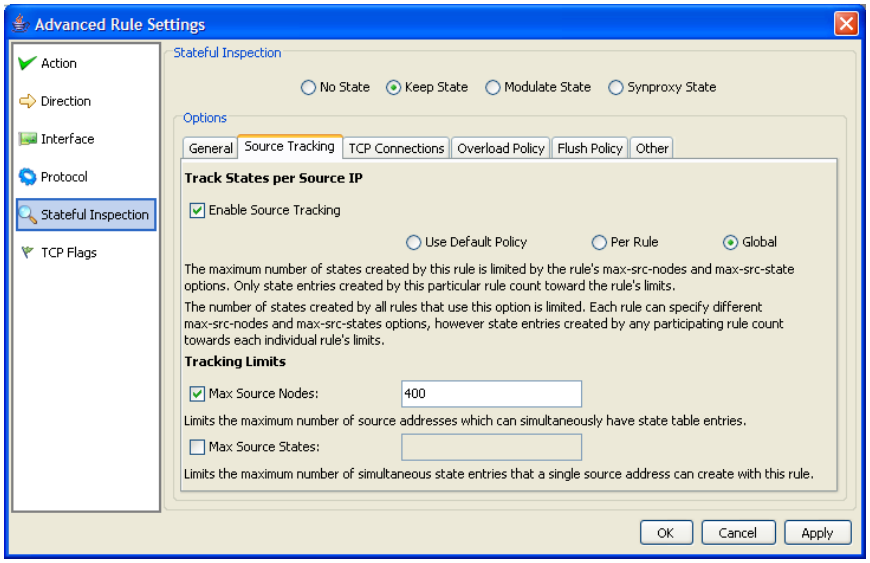

Figure 4.12: Keep State Source Tracking.

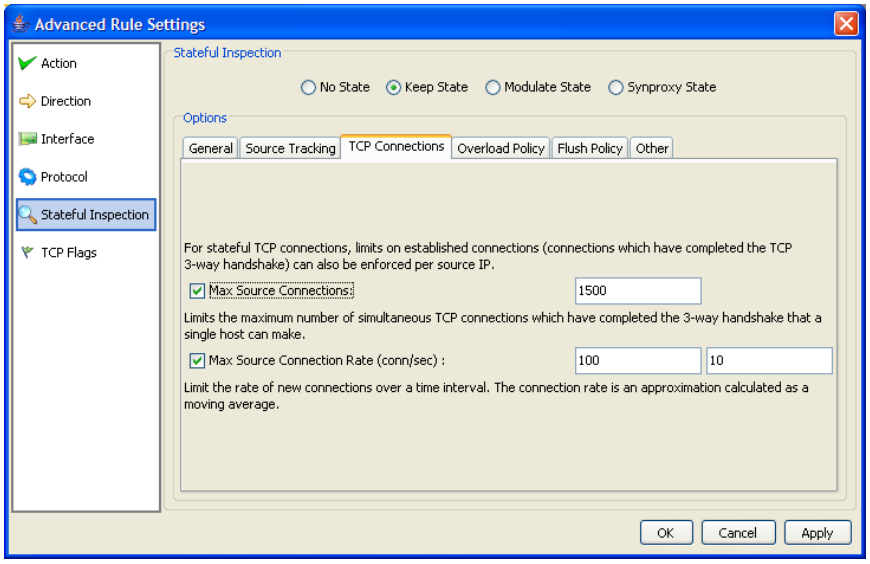

Figure 4.13: Keep State TCP Connection Settings.

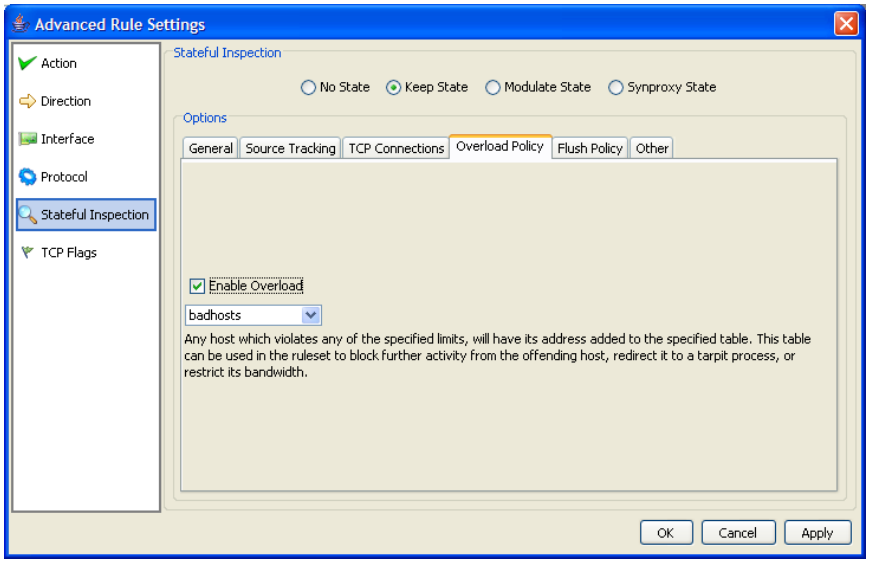

Figure 4.14: Keep State Overload Policy.

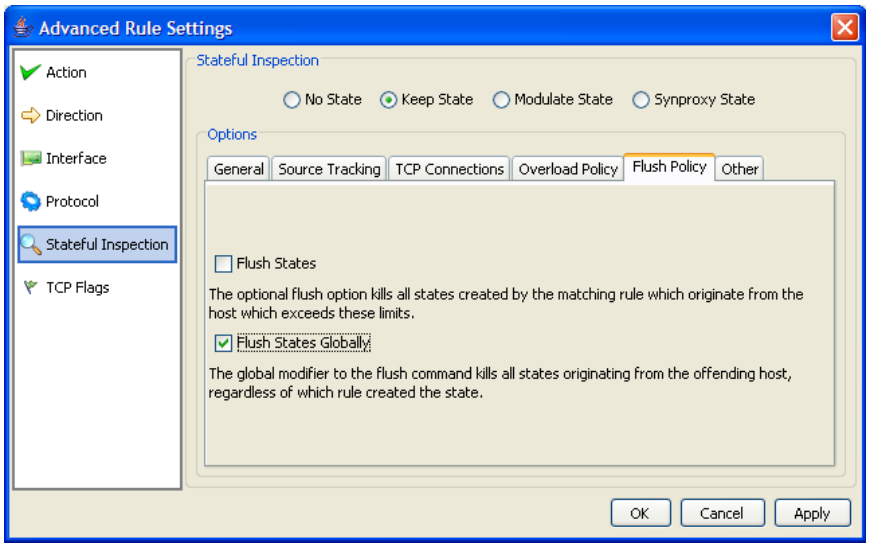

Figure 4.15: Keep State Flush Policy.

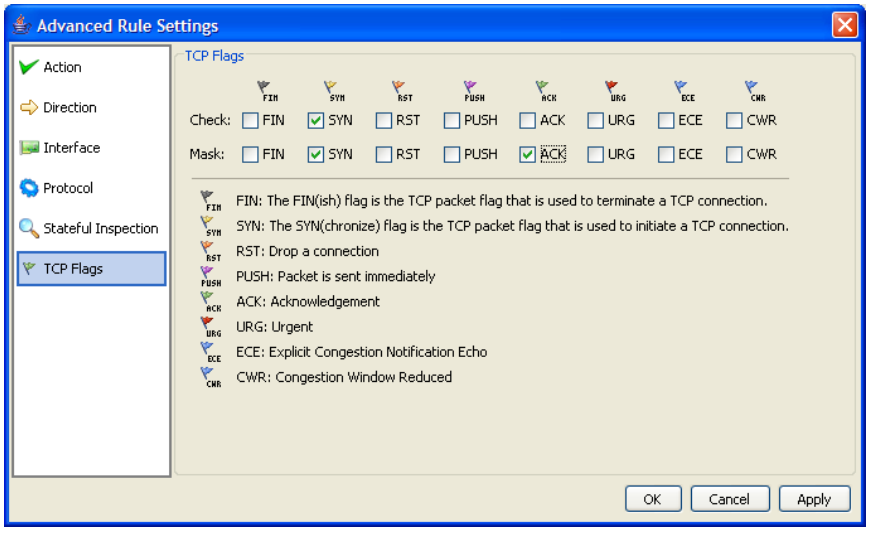

Figure 4.16: TCP Flags Settings.

Chapter 5

Testing

As this project has focused primarily on the GMC part of the planned system, and minimal work has been performed on the other two parts, it is not relevant to talk about system testing of the CMP or CAS. A total integration test plan has not been developed. The only element of internal system communication which is clear, is that SSH2 subsystems and SCP will be used for data transfer. Reasonably complete is also how the preliminary protocol for Job specification looks like. All in all, the only part of the system that has been tested in any sense is the GMC.

In starting the planning and prototyping phases there was little awareness of the need to select a testing method. No real planning was done in regards to how testing would be performed. Though the development model specified has been followed, with an unit based iterative cycle of planning and development, the actual focus on unit testing has been minimal. The work has been seen as a prototyping effort, more than a focus on commercial quality code. This has resulted in reducing testing to the mere necessary act of getting code to compile correctly, and to perform the requested task correctly. This method of work is quite common in unstructured development projects, and may be compared to a more «code-and-fix» way of developing, though the way each unit or feature has been functionally tested is more in the line of a white-box test.

Having said this, it must be pointed out, that each time a major feature has been introduced into the system, a more thorough test has been performed to verify this features function and integration. Such test-points have occurred for instance with the introduction of

- the collapsible/expandable panels
- navigational buttons and tabbed panes in filter editor
- drag-and-drop functionality
- command and action interfaces and classes
- the Castor generated data model for PF

- the advanced rule settings dialog
- the core GUI framework

At these points, we have used the white-box testing paradigm to verify that the new features work as intended, and that they do not break other parts of the application. White-box testing is a concept name for structural testing. This is performed by examining code, and subjecting the product to test data and usage patterns, where the test cases are derived from transparent knowledge of how the code works[16]. This method of testing is practically the opposite of black-box testing, where test cases are designed with basis in expected functionality, not in knowledge of actual construction, and the test cases are fed through the application or system, checking that the end result is as expected. The system is viewed as a black box, for which you know what to put in to, and have an expectation to the outcome.

The Castor library has been used as parser and generator for the XML files that describe PF configurations and the hierarchy of managed firewalls. In order to ensure that the Java data model is completely compatible with the Castor parser/generator, the source code generator included in the Castor package has been used to auto-generate the Java code for these data models. As a method to test this auto-generated code, simple unit test Java classes have been created, that create test configurations within the Java virtual machine, and perform black-box unit testing on the model and both Castor XML marshalling and unmarshalling[1]. The resulting XML files have been read through to verify that the marshalling process completes correctly, and the debugging features in Eclipse have been used to check that in-memory representation of data read in from XML is correct. The XML files generated by this black-box process has also been used as basis for prototype testing, and for white-box testing of application features.

It is worth noting that at regular intervals, we have had assistance from the group that we have shared workspace with, in testing the user interface. These tests have been performed in a black-box way, where the person who has assisted has been given a rough explanation of what the application should be able to do, before he has been given full freedom in exploring the application interface. This has proven to be an efficient way to uncover flaws that we had not expected to appear while doing our white-box tests.

Methods such as equivalence partitioning and path testing have not been applied in our defect testing, and the reason for this may be explained in the fact that we have not had a clear strategy for testing.

Should we try to find a principle for integration testing that is similar to the informal approach we have made, this will have to be the bottom-up approach, where low level components are integrated and tested before higher level components have been developed [16].

[1] In computer programming, the marshaller converts data parameters from procedure calls into a standardized data structure for transfer or storage. This data is later decoded or «unmarshalled» by a receiver.

When it comes to security testing, this have to be done when the application is closer to release. We have only done prototyping so far, so we have seen no point in doing security tests yet. Before the application goes in release a thorough security test case must be in place, testing for how vulnerable the system is for things like DoS, Man-In-The-Middle attacks and key tampering attempts [17].

Chapter 6

Discussion of results

6.1 Introduction

A discussion of results often tend to focus on what has not been completed, what has been done wrong, and similar problems. This text is no exception. But before starting the summary of what is missing, or could have been done differently, it is necessary to note that at the end of the project, all members of the project group feels that we have completed the task we took upon us. What we promised to deliver, was a foundation of a management solution, through documented design, requirements definition, and some prototype code. By reviewing the requirements specification and design documents, we feel it can be claimed without problem that the design and concept project goals have been reached. Also, by examining the developed application prototype, and implementation description, we feel that we can claim to have completed the primary goal; to introduce a different and functional concept for visually editing firewall rules.

6.2 Deviations from requirements

6.2.1 Management server and firewall configuration software

From the very beginning of the project period, there was a clear plan to perform more development of all parts of the system, as compared to what we have achieved. Though time was allocated to development of prototype development on the management server and firewall configuration software, we ended up focusing more on the GUI application, thus taking away the development time allotted to the other parts of the system. The reason for this will be discussed further down.

When it comes to the planning and design of the management server and configuration software, there are a few elements to point out. First of all, we must admit to have discovered a flaw in the system construction that has followed us throughout the entire project from inception. The Copwall system aims to become a tool for configuration of firewalls on multiple platforms, but relies on installing

system software on the firewalls that are to be managed. There is no problem to develop and install this software as an OpenSSH subsystem on a computer running OpenBSD, but it may not be possible at all to install user defined software on other platforms. In addition to this, the concept of configuration software installed on the managed firewalls, complicates both installation of the system, and the construction of the overall system. By eliminating the named Configuration and Administration System completely, by using direct command shell interaction from management server and/or GUI application, it may become a smaller task to support the two modes of operation, as server-based and standalone configuration tool.

We have also, in the design document, completely forgotten one of the important concepts of the original problem definition in regards to the management server. It is noted quite early that a modular construction should be used in the management system, that separates functions related to communication with, and configuration of firewalls, into separate modules for different operating systems and/or firewall platforms. As is clear when examining the design document, is that this has sometime during the project been overlooked, as no such platform dependent module structure exists. Trying to find a reason for this, one has to conclude that the major factor must be that we have focused completely at OpenBSD as the platform to be configured.

6.2.2 Graphical management application

In addition to the focus on OpenBSD, there has been a clear focus on the GUI application, and hence the GUI configuration tool. This part of the system is a highly central element of the original project definition, and is seen as the core functionality of the application. As such, it was decided that we should focus on developing as much as possible of a functional filter configuration tool. Seeing that GUI application prototyping and development is a large task, it was decided that software development of the other sections should be postponed. System software development of the type needed for the CMP and CAS has not been dismissed as a minimal job, but we see this as more a job of using our existing knowledge of the Unix operating system family and C programming, thereby making the development of CMD and CAS more a job of making the elements of the system work together, rather than creating something completely new.

In the GMC, we have utilized the Castor XML framework/library. This gave us a large advantage, by being able to auto-generating Java source code from a XML Schema Definition. By building a precise XSD representation of the PF configuration format, Castor helped us get an efficient Java representation of a very complex data structure, with integral XML support. But as a famous saying goes, «There's no such thing as a free lunch». Gaining advantages with Castor, locked down our data model, preventing the addition of user defined methods to the objects generated by Castor. Using a GUI and underlying controllers to manipulate the data has at times become a complex and confusing task. This has at times slowed

progress quite drastically, trying to find solutions that work.

It was an intention from both project group and employer, to use a true pluggable architecture on top of the basic functionality of the management application. This has not been achieved. We have however, tried to get a modular structure of the application. To some extent, this has been achieved, although some critic may be raised also here. Firstly, there has been written very scarce documentation in regards to software interfaces and module loading methods. Secondly, the modular structure is currently geared towards supporting a single type of firewall. Again we see that we should have had a higher awareness of the planned multi platform support, along with a higher awareness of developing concrete interfaces with accompanying documentation.

6.2.3 Configuration generator

Though we have a complete data structure implemented to represent filter configurations, and support of storing the configurations to XML file format, we have not started work on the system to translate XML configurations to the configuration format expected by the PF software on firewalls. The same goes for the reverse parser, for translating configuration file format in to the data model and XML format.

6.2.4 Summary of deviations

The following list is a summary of what elements defined in the requirement specification that remains unimplemented in software:

- CMP is completely unimplemented, but is documented and designed.
- CAS is completely unimplemented, but documented and designed.
- Firewall browser in GMC is mocked up, but not implemented.
- Interface configuration, service configurators, log viewer, and system configurator have gone though some prototyping, and some design has been performed, but no code has written.
- Configuration file generator/parser, job request system and SSH communication modules in the GMC are designed and described, but no actual code has been created.
- The PF editor has not been completed, though the central concepts are demonstrated, and complete its specified requirements.
- The implementation of a modular structure in the application is not as rigid and well documented as it should have been.

6.3 Other considerations

We see that there is a risk involved in introducing a graphical configuration solution of the type we have created. It is difficult to assess how the user community will react to a solution that is completely different from what is accepted as a norm. In addition to this, none of the members of the project group have any formal knowledge of ergonomic concepts. We have stated more than once, that our solution is supposed to have an interface that is easy to understand and use. But we have no basis to debate this in form of a human-computer interface principle discussion, as none of us have knowledge within this field.

To our defence, we have had external users test our application. These users have been a mix of users that are very experienced with PF configurations, and users who are completely new to the principles of firewalls. From these test users, the responses have been that the filter editor has a logical construction, and represents in a good way how a filter setup does work in reality. We can also through the feedback conclude that the application does implement its set of the requirements.

The development of the GMC has been based to great extent on throw-away prototyping. This has sometimes resulted in having good technical solutions introduced late in the process. As a result, some sections of the application has been completely rewritten several times. Some of the solutions used have also been the result of lengthy research and test prototyping before actual implementation. It is agreed within the group that the method of throw-away prototyping has at times used up too much of our available time.

6.4 Potential for completion

The following is an ordered list of what needs to be done to make the system fully operational, as described in requirements specification. It is worth noting again that our problem definition clearly states that we have never expected to get the system fully operational.

1. make filter configuration module complete
2. implement firewall management browser module
3. implement module for network interface configuration
4. implement missing features of core user interface
5. implement services configuration module
6. implement system configuration module
7. implement configuration generator and parser
8. implement SSH communication in GMC

9. get direct communication between GMC and firewall operational
10. implement status and logging module
11. implement the CMP, and introduce use of this into the GMC.

At the points 7, 9 and 11 release milestones should be set. At these points the system should have enough functionality to represent a useful tool that can be released to the public. The first release, at point 7, should allow all forms of configuration to be done, using local file import and export, and manual copying to and from firewalls. At second release, point 9, the system is complete as a standalone configuration editor, which supports import and export of configurations directly via SSH to firewalls. The third release is the complete system release, where all features are implemented. From this release on, work should be done to bring the application to a stable and tested state, ready for release of version 1.0.0.

6.5 Potential for expansion

After the release of the stable version 1.0.0, there is still room for expansion. First of all, it should be a small task to implement the more traditional tables based filter rule editor. It is also very interesting to expand the graphical filter editor with a resource editor, to create tables, macros and similar structures that simplify a PF configuration, and making this resource editor a source for drag-and-drop of such elements onto the graphical editor view.

The system should also be rewritten to use a true plug-in structure, followed by serious work on supporting multiple firewall platforms, like Linux IP-Tables, FreeBSD IPFW and Cisco PIX.

It is also very interesting to introduce a third view on filter rule editing. Most system administrators who have worked with OpenBSD PF over some time, have developed a fair knowledge of the configuration file syntax. These professionals may want a text view of the configuration. At the end of the project period, suggestions were made to include a syntax highlighting editor, featuring code completion, and syntax verification. Since the system already should contain a solution for parsing configuration files into the application data representation, this should not be impossible, using inspiration from the way Eclipse editors are constructed.

Chapter 7

Conclusion

7.1 Evaluation of the project task

As the group was responsible for defining the initial project description, one may claim that we are in no position to evaluate whether we chose the correct task for our graduation project. But the problem definition that we ended up with is very different from the initial description. As an evaluation of the project size we see that we perhaps should have started setting restrictions to expanding ideas presented to us from our employer, at an earlier stage. But the ideas presented were all very interesting, and we were at times quick to embrace these ideas. The resulting system design and specification has a broader market interest than the rather small project that we initially suggested.

We feel that the problem we set out to tackle was the one that gave us the most, in form of challenges and personal engagement. We also feel that we have accomplished what was required of us.

7.2 Additional gains

During this project, we have in addition to testing and expanding on the basis that our education provides, gotten to learn about quite a few useful tools and techniques. Our knowledge on Java as an application programming language has been greatly expanded, along with new insight in good Java programming techniques like the model / view / controller paradigm, and how to efficiently use external libraries. In a project management setting, we have not only examined new methodologies not known to us through previous education, but actually introduced a new model ourselves. This model we feel that has worked perfectly for us, and we feel it may be applied also to other projects.

One area where much has been learned, which is not directly related to educational learning, is in tools that have been used. We have located several tools that have had a definitive impact through increasing productivity. Perhaps the most central of these is the Eclipse platform, an extensible integrated development envi-

ronment that also provides software development kits that the user may use in his or her own applications. Eclipse provides a host of features that for us significantly speeded up Java development compared to other environments and solutions we have tried. Comparisons include Borland JBuilder, NetBeans, SunOne Studio and developing outside of an IDE.

Another tool that was introduced to the project was LaTeX2$_\epsilon$. This was introduced as a requirement by Jon Langseth, to be used for all documentation except figures and source code comments. LaTeX2$_\epsilon$ was at the time completely unknown to the rest of the project group, and initially had a rather steep learning curve. All members are now comfortable with this format. Though some elements still may feel cumbersome, the general consensus is that LaTeX2$_\epsilon$ is in invaluable tool for documentation and report generation.

7.3 Evaluation of project as form of work

The graduation project is the culmination of three years of study, and as such, should test as many aspects of our education to date as possible. It is hard to find any other way to actually test that students have learned what they need to, and also understood how to apply this knowledge in a real work situation. A large project at the end of the education is the closest one may come to simulating a real working environment, while maintaining the control and mentoring needed by an educational institution.

As we now round off five months of project work, we feel that we have gotten a good result from working on a project basis. Our knowledge has been tested, and we have gained significant understanding of how to apply what we have learnt in our study in practical use. We have also gotten a few eye openers in regards to the need for documentation, system design, planning and testing.

Bibliography

[1] Jacek Artymiac. *Building Firewalls with OpenBSD and PF*. devGuide.net Artymiac, USA, second edition, 2003.

[2] The OpenBSD project. Openbsd release song lyrics, openbsd 3.5. http://www.openbsd.org/lyrics.html#35.

[3] Federico Biancuzzi. Openbsd pf developer interview, part 2. http://www.onlamp.com/pub/a/bsd/2004/05/06/pf_developers.html.

[4] Nebulon Pty. Ltd. Feature driven development web site. http://www.featuredrivendevelopment.com/.

[5] Control Chaos. Scrum, it's about common sence. http://www.controlchaos.com/.

[6] Linus Torvalds. Linux kernel release 2.0.xx readme. http://www.kernel.org/pub/linux/kernel/README.

[7] Paul C. Kocher Alan O. Freier, Philip Karlton. *The SSL Protocol Version 3.0*. Alan O. Freier, Philip Karlton, Paul C. Kocher, Transport Layer Security Working Group, http://wp.netscape.com/eng/ssl3/draft302.txt, November 1996.

[8] Mark Burgess. *Principles of network and system administration*. John Wiley & Sons Ltd, West Sussex, England, second edition, 2004.

[9] Craig Larman. *Applying UML and patterns : an introduction to object oriented analysis and design and the Unified Process*. Prentice Hall PTR, Upper Saddle River, NJ, USA, second edition, 2002.

[10] Wolf Paulus. Swixml web site. http://www.swixml.org.

[11] Sun Microsystems, Inc, http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html. *Code Conventions for the Java^{TM} Programming Language*, 1996.

[12] Guy Steele James Gosling, Bill Joy and Gilad Bracha. *Java Language Specification*. ADDISON-WESLEY, California, U.S.A., third edition, 2005.

[13] Dennis M. Sosnoski. Xml and java technologies: Data binding with castor. *IBM developerWorks*, April 2002. http://www-106.ibm.com/developerworks/xml/library/x-bindcastor/ (18.feb, 2005).

[14] The OpenBSD project. Pf: The openbsd packet filter. http://www.openbsd.org/faq/pf/index.html.

[15] The OpenBSD project. Openbsd 3.6 pf.conf manual page. http://resin.csoft.net/cgi-bin/man.cgi?section=5&topic=pf.conf.

[16] Ian Sommerville. *Software Engineering*. Pearson Education Ltd, Essex, England, sixth edition, 2001.

[17] Inc. Cisco Systems. Internetworking technology handbook, security technologies. http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/security.htm.

Appendix A

Glossary

A

- **API** - Application Programming Interface

B

- **BiNAT** - Bi-directional NAT
- **Black-box** - A testing method where the system is wieved as an opaque, black box.
- **BNF** - Backus-Naur form (also known as Backus normal form)
- **BSD** - Berkeley Software Distribution
- **BSD family** - The family of BSD* operating systems (Open, Free, Net and others)
- **Bug** - An insect, or a flaw in programming code.
- **Buggy code** - Code which contains bugs.

C

- **CARP** - Common Address Redundancy Protocol
- **CAS** - Configuration and Administration System
- **Castor** - An Open Source data binding framework for Java
- **C/C++** - A programming language
- **CDT** - C/C++ Development Tool
- **Cisco** - A major actor in the networking hardware market
- **Cisco PIX** - A firewall hardware family and standard by the Cizzz-coeee company
- **CMP** - Central Management Platform
- **Copwall** - The name Copwall originates in an acronym representing CARP, OpenBSD and PF, followed by the word wall.
- **CoreUI** - Core User Interface in Copwall
- **CVS** - Concurrent Versions System, https://www.cvshome.org/

D

- **Daemon** - A process which runs in the background and provides some service to an application, on behalf of a user.
- **DoS attack** - Denial of Service attack
- **DSH** - Data Store and revision Handling system
- **DTD** - Document Type Definition

E

- **Eclipse Platform** - Integrated Extensible Development Environment, see http://www.eclipse.org
- **ECTS** - European Credit Transfer System
- **EMF** - Eclipse Modelling Framework

F

- **FDD** - Feature Driven Development
- **Firewall** - A computer acting as an interface between two networks (e.g., the Internet and an private network, respectively), and regulates traffic between those networks for the purpose of protecting the internal network from electronic attacks originating from the external network
- **FLG** - Firewall Logical Group
- **FLU** - Firewall Logical Unit
- **FPU** - Firewall Physical Unit

G

- **Gantt chart** - A popular type of bar chart, showing the interrelationships of how projects, schedules, and other time-related systems progress over time
- **GMC** - Graphical Management Client. Aka. Copwall
- **GMCI** - Graphical Management Client Interface
- **GNU** - GNU Not UNIX
- **GUC** - Gjøvik University College
- **GUI** - Graphical User Interface

H

- **HSRP** - Hot Standby Router Protocol,
  see http://www.cisco.com/en/US/tech/tk648/tk362/tk321/tech_protocol_home.html

I

- **ICMP** - Internet Control Message Protocol
- **IDE** - Integrated Development Environment
- **IETF** - Internet Engineering Task Force
- **IP-tables** - The implementation of TCP/IP filtering and inspection of Linux 2.4 and newer.
- **IPFW** - IP FireWall, the FreeBSD implementation of TCP/IP filtering.
- **IPv4** - Internet Protocol version 4
- **IPv6** - Internet Protocol version 6
- **ISN** - Initial Sequence Number
- **ISO-8859-1** - Standard encoding for latin based and Western European characters, see http://en.wikipedia.org/wiki/ISO_8859-1

J

- **J2SE** - Sun Java 2 Platform Standard Edition
- **JAR** - A file format developed by Sun to store compressed Java programs
- **Java** - Programming Language
- **JavaDoc** - A documentation tool for Java
- **JDT** - Java Development Tools
- **JM** - Job Manager
- **JPF** - Java Plugin Framework
- **JRE** - Sun Java Runtime Environment
- **JS** - Job Scheduler
- **JSCh** - Java Secure Channel, a SSH2 library for Java

L

- **LAN** - Local Area Network
- **LaTeX2e** - LaTeX is a high-quality typesetting system, with features designed for the production of technical and scientific documentation. LaTeX is the de facto standard for the communication and publication of scientific documents. See http://www.latex-project.org/
- **libssh** - SSH client library for C/C++, http://www.0xbadc0de.be/?part=libssh

M

- **Man-In-The-Middle attack** - an attack in which an attacker is able to read, insert and modify at will, messages between two parties without either party knowing that the link between them has been compromised
- **Marshalling** - In computer programming, the marshaller converts data parameters from procedure calls into a standardized data structure for transfer or storage. This data is later decoded or «unmarshalled» by a receiver
- **MVC** - Model / View / Controller Paradigm

N

- **NAT** - Network Address Translation

O

- **OpenBSD** - A FREE, multi-platform 4.4BSD-based UNIX-like operating system. Emphasizes portability, standardization, correctness, proactive security and integrated cryptography. See http://www.openbsd.org for more information.
- **OpenSSH** - The OpenBSD project's implementation of the SSH protocol family, http://openssh.org/
- **ORO** - A set of text-processing Java classes developed by the Apache Jakarta project

P

- **Patch** - An addition to a piece of code to remove an existing bug or misfeature
- **PDE** - Plugin Development Enviroment
- **Pentium™** - A family of microprocessors produced and marketed by Intel®

- **Perl** - A scripting language, created by Larry Wall. http://www.perl.com/
- **PF** - OpenBSD's filtering engine: «Packet Filter». Originally coded by Daniel Hartmeier
- **pf-rule** - A PF rule of type; filter, normalization, antispoof, NAT, BINAT, redirect or queue rule
- **pf.conf** - The default configuration file for PF.
- **PostgreSQL** - PostgreSQL is a highly scalable, SQL compliant, open source object-relational database management system, see http://www.postgresql.org/
- **Python** - A scripting language, created by Guido van Rossum. See http://www.python.org/

Q

- **QoS** - Quality of Service

R

- **RCP** - Rich Client Platform
- **Regexp** - Regular Exp
- **RSA** - Encryption algorithm named after the three Mathematichians inventing it; Ronald Rivest, Adi Shamir og Leonard Adleman

S

- **SCA** - System Configuration Agent
- **SCP** - Secure CoPy, utilizes the SSH protocol for secure file copy
- **SCRUM** - Scrum is an agile, lightweight process that can be used to manage and control software and product development using iterative, incremental practices.
- **SDK** - Software Development Kit
- **SFTP** - Secure FTP, utilizes the SSH protocol for secure FTP sessions
- **SOHO** - Small Office or Home Office
- **SQL** - Structured Query Language
- **SSH** - Secure Shell
- **SSHd** - SSH daemon, see daemon

- **SSL** - Secure Sockets Layer
- **STDIN** - Standard Input stream
- **STDOUT** - Standard Input stream
- **STDERR** - Standard Error stream
- **Swing** - Provides a set of «lightweight» components that, to the maximum degree possible, work the same on all platforms
- **SWT** - Standard Widget Toolkit
- **SYN flood** - The SYN flood attack sends TCP connections requests faster than a machine can process them
- **syslog** - A facility in UNIX and UNIX-workalike systems which controls the logging of system events

T

- **TCP** - Transmission Control Protocol

U

- **UDP** - User Datagram Protocol
- **UML** - Unvirsal Modeling Language
- **UNIX** - It's more than an operating system! It's a religion!
- **UNIX pipes** - Simply put, a pipe is a method of connecting the standard output of one process to the standard input of another.
- **Unmarshalling** - See Marshalling
- **User** - The user of the Copwall application

V

- **VC** - View / Controller
- **VM** - Virtual Machine
- **VLAN** - Virtual LAN
- **VPN** - Virtual Private Network
- **VRRP** - Virtual Router Redundancy Protocol, see ftp://ftp.rfc-editor.org/in-notes/rfc3768.txt

W

- **WAN** - Wide Area Network
- **White-box** - A testing method where the system is wieved as a fully transparent box.

X

- **X.509 certificate** - A widely used standard for defining digital certificates
- **Xerces-J** - a fully conforming XML Schema processor
- **XML** - eXtensible Markup Language
- **XSD** - XML Schema Definition