

Final Report

# tigaNAV

A NAV development project



*“Towards NAV v3”*

**Project leader**

Vidar Faltinsen

**Project group**

John Magne Bredal – Kristian Eide – Sigurd Gartmann  
Bjørn Ove Grøtan – Hans Jørgen Hoel – Erlend Mjåvatten  
Magnus Thanem Nordseth – Andreas Åkre Solberg  
Magnar Sveen – Stian Søiland – Gro-Anita Vindheim  
Morten Vold – Arne Øslebø

*4<sup>th</sup> of December, 2003*



ITEA, NTNU



## Abstract

tigaNAV is a NAV development project. NAV is short for Network Administration Visualized and is NTNU's self developed network management solution. Development has been going on for five years, the last three with support from UNINETT. Currently NAV version 2.1.5 is available, and up and running on all four Norwegian universities and 10 colleges.<sup>1</sup>

The focus in tigaNAV has been on NAV version 3. Actually, this work started prior to tigaNAV, in the summer of 2002, with project NAVMore<sup>2</sup>. tigaNAV has continued in the footsteps of NAVMore, the timeframe has been June to November 2003. A lot of resources have been put into the project, 14 developers have been involved and more than 3500 work hours are spent.

tigaNAV has produced many results, this report summarizes the work. The main objective has been to complete the development of NAV v3 with all its subsystems. At the time of writing we are entering an alpha test period of NAV v3 at NTNU. Beta-testing is planned for February to April next year and will include installations at UiTø and HiMolde. The projected release date for NAV v3 is set for April 2, 2004. Within this timeframe we will finish off unfinished work and bug fix existing code. This aside, there will be a complete feature freeze of the system until the summer of 2004.

The focus in tigaNAV has been twofold. First we have looked at NAV as a software product and made many remarkable improvements (this aspect of NAV has not really been addressed before). Second we have improved existing functionality and introduced new features. The table on page 87 of this report gives an at-a-glance overview on the status of all aspects of NAV.

tigaNAV has introduced a completely new and consistent user interface based on a common programming language (python) with the use of templating. We have adopted a general and powerful authentication and

---

<sup>1</sup> As of now there is no publicly available licence of NAV. NAV is owned by NTNU. UNINETT has through project participation gained free access to the product, for itself and its members.

<sup>2</sup> Many NAVMore results are included in 2.1.5. Other results have been in use at NTNU throughout this year.

authorization mechanism. Subversion has replaced CVS as our proven version control system. A modular software build system replaces the existing monolithic approach.

Under the hood we have done a complete rewrite of the data collection system. We have improved NAVdb to even more accurately model the running network. The data collection system now uses a plug-in based architecture with fine-grained control of collection intervals, working in parallel, and basing its SNMP collection on a central OID database. A semi-automatic type classifier is included improving NAV's ability to support new network equipment. We have also replaced the NAV v2 seed files with a superior web front end.

The functional improvements in tigaNAV include:

- A more general operational status page with status on all operational events (eventually). An integrated message system to inform NAV users of scheduled outage, special faults and other operational events.
- The device browser presenting all information on a device in one web page with links to reports, statistics, switch port data etc.
- The network explorer introducing a graphical display of the network layout on a per vlan basis.
- An RRD browser with the ability to gather different statistics on the same page or even in the same graph.
- Device management with the ability to track milestone events of physical devices from order and arrival through the operational stages.

With regard to progress, the project plan was too optimistic. The release of NAV v3 is delayed by 4 months. There are certainly many challenges in a project of this size, this is elaborated on in the report.

To conclude, the project leader is very pleased with the amount of work done – and the quality. There is a fantastic enthusiasm surrounding NAV. The tigaNAV team is a group of highly skilled developers who believe in what they do and do their very best.

It remains to be seen if tigaNAV has achieved its goal. 2004 will reveal if NAV v3 is a proven solution. We certainly think so.

# Table of contents

<b>PREFACE .....</b>	<b>9</b>
<b>1. INTRODUCTION .....</b>	<b>10</b>
1.1 WHAT INITIATED THIS PROJECT .....	10
1.2 THE PROJECT OBJECTIVE .....	10
1.3 EFFECTS OF THE PROJECT RESULTS.....	11
1.4 PERSONNEL RESOURCES .....	11
1.5 SUBPROJECTS AND WORK HOURS .....	11
1.6 OUTLINE OF THIS REPORT .....	13
1.7 NAV GLOSSARY .....	13
<b>2. AUTHENTICATION, AUTHORIZATION, PROFILES.....</b>	<b>14</b>
2.1 RESOURCES .....	14
2.2 MAIN OBJECTIVE .....	14
2.3 RESULTS.....	14
2.3.1 User database .....	14
2.3.2 Authentication and authorization .....	14
2.3.3 User admin panel.....	17
2.3.4 Sessions.....	18
2.4 PROBLEMS.....	18
2.4.1 Double login .....	18
2.4.2 Strange session related error messages .....	19
2.5 FURTHER WORK .....	19
2.6 CONCLUDING REMARKS .....	19
<b>3. THE USER INTERFACE .....</b>	<b>20</b>
3.1 RESOURCES .....	20
3.2 MAIN OBJECTIVE .....	20
3.3 RESULTS.....	20
3.3.1 Templating solution .....	20
3.3.2 Web interface.....	21
3.3.3 Front page .....	22
3.3.4 Toolbox .....	23
3.3.5 Quick link preferences .....	24
3.4 PROBLEMS.....	25
3.5 FURTHER WORK.....	25
<b>4. EVENT AND ALERT SYSTEM .....</b>	<b>26</b>
4.1 RESOURCES .....	26
4.2 MAIN OBJECTIVE .....	26
4.3 RESULTS.....	26
4.3.1 Event Engine.....	26
4.3.2 Alert Engine.....	28
4.3.3 The SMS daemon .....	28
4.3.4 Alert Profiles.....	28
4.4 PROBLEMS.....	31
4.5 FURTHER WORK .....	31
4.5.1 Event Engine.....	31
4.5.2 NAV Profiles .....	31

<b>5: THE NAV V3 DATA COLLECTION SYSTEM .....</b>	<b>32</b>
5.1 RESOURCES .....	32
5.2 MAIN OBJECTIVE .....	32
5.3 DATA COLLECTION IN NAV V3 AT A GLANCE .....	32
5.4 SUBSYSTEM OVERVIEW .....	33
5.4.1 NAVdb updates .....	33
5.4.2 <i>getDeviceData</i> .....	33
5.4.3 <i>The OID database</i> .....	33
5.4.4 <i>GetDeviceData Plugins</i> .....	33
5.4.5 <i>The cam logger</i> .....	34
5.4.6 <i>Network topology discovery</i> .....	34
5.4.7 <i>Vlan discovery</i> .....	34
5.5 DETAILED DESCRIPTION OF NEW SUBSYSTEMS .....	34
5.5.1 <i>The NAVdb</i> .....	34
5.5.2 <i>getDeviceData</i> .....	39
5.5.3 <i>The cam logger</i> .....	43
5.5.4 <i>Network topology discovery</i> .....	44
5.5.5 <i>Vlan discovery</i> .....	44
5.6 PROBLEMS .....	45
5.7 FURTHER WORK .....	45
5.7 CONCLUDING REMARKS .....	46
<b>6. NEW FRONT END SUBSYSTEMS .....</b>	<b>48</b>
6.1 RESOURCES .....	48
6.2 MAIN OBJECTIVE .....	48
6.3 RESULTS .....	48
6.3.1 <i>editDB: A web based front end to NAVdb</i> .....	48
6.3.2 <i>The Status page</i> .....	53
6.3.3 <i>Device Management</i> .....	55
6.4 PROBLEMS .....	58
6.5 FURTHER WORK .....	58
<b>7: RRD ACTIVITIES .....</b>	<b>59</b>
7.1 MAIN OBJECTIVE .....	59
7.2 RESULTS .....	59
7.2.1 <i>RRD database</i> .....	59
7.2.2 <i>makecricketconfig</i> .....	60
7.2.3 <i>A better and more flexible way to view graphs</i> .....	61
7.2.4 <i>Sorted Statistics for all RRD data</i> .....	62
7.2.5 <i>Threshold Monitor</i> .....	62
7.2.6 <i>Large Scale Cricket Test</i> .....	63
7.2.7 <i>Reliable Collection of Data to the network load map</i> .....	64
7.3 PROBLEMS .....	64
7.4 FURTHER WORK .....	64
<b>8. ENHANCED MESSAGE OF THE DAY .....</b>	<b>65</b>
8.1 RESOURCES .....	65
8.2 MAIN OBJECTIVE .....	65
8.3 RESULTS .....	65
8.3.1 <i>Message of the day</i> .....	65

8.3.2 <i>Set on maintenance</i> .....	66
8.3.3 <i>Database design</i> .....	67
8.3.4 <i>Background processes</i> .....	67
8.4 FURTHER WORK .....	68
<b>9. DEVICE BROWSER.....</b>	<b>69</b>
9.1 RESOURCES .....	69
9.2 MAIN OBJECTIVE .....	69
9.3 RESULTS.....	69
9.3.1 <i>General view</i> .....	69
9.3.2 <i>Services</i> .....	70
9.3.3 <i>Modules and Ports</i> .....	71
9.3.4 <i>Proper URLs</i> .....	72
9.3.5 <i>RRD browser</i> .....	72
9.4 PROBLEMS.....	73
9.4 FURTHER WORK .....	74
9.5 CONCLUDING REMARKS .....	74
<b>10. ROUND TRIP AND PACKET LOSS .....</b>	<b>76</b>
10.1 RESOURCES .....	76
10.2 MAIN OBJECTIVE .....	76
10.3 RESULTS.....	76
<b>11. NEW NETWORK UTILITIES.....</b>	<b>78</b>
11.1 RESOURCES .....	78
11.2 MAIN OBJECTIVE .....	78
11.3 RESULTS.....	78
11.3.1 <i>Machines behind a switch port</i> .....	78
11.3.2 <i>Recently used switch ports and on-the-fly status</i> .....	78
11.3.3 <i>Network Explorer</i> .....	79
<b>12. VERSION CONTROL AND SOFTWARE BUILD .....</b>	<b>81</b>
12.1 RESOURCES .....	81
12.2 MAIN OBJECTIVE .....	81
12.3 RESULTS.....	81
12.3.1 <i>CVS vs. Subversion</i> .....	81
12.3.2 <i>Restructuring source code repository layout</i> .....	82
12.3.3 <i>Software build system</i> .....	82
12.3.4 <i>Restructuring installation layout</i> .....	83
12.4 PROBLEMS.....	84
12.5 FURTHER WORK .....	85
12.6 CONCLUDING REMARKS .....	85
<b>13. NAV VERSION 3 .....</b>	<b>86</b>
13.1 SUMMARY OF NAV v3 FEATURES .....	86
13.2 NAV v3 TEST AND RELEASE PLAN .....	88
<b>14. SUMMARY AND CONCLUSION.....</b>	<b>90</b>
14.1 PROJECT SELF CRITICISM .....	90
14.2 CONCLUSION .....	92

## List of figures

<i>No</i>	<i>Legend</i>	<i>Page</i>
1	How Cheetah works	21
2	The web interface	22
3	A front page with no eMotD or boxes down	23
4	The quick link preferences page	24
5	The Event and Alert System	27
6	The NAV Profiles database	29
7	The Alert Profile main page	30
8	NAVdb with new and/or significant fields	35
9	GetDeviceDate component overview	39
10	The edit database subsystem	49
11	Web form for adding a new netbox (IP device)	50
12	Form for editing or deleting equipment types	50
13	Flow diagram for adding a IP device	52
14	The Status Page	54
15	User Preferences for the Status Page	54
16	The device triangle: device, module and netbox	55
17	Processes in a device's life	56
18	Life cycle of two physical devices	57
19	The RRD database	60
20	Example of eMotD message	66
21	The eMOTD database design	67
22	Inconsistent outage and maintenance timeframe	68
23	The Device Browser	70
24	Services view in device browser	70
25	The services matrix	71
26	The Device Browser Switch View	72
27	The RRD browser	73
28	Packet loss and roundtrip time	77
29	Service (imap) availability and response time	77
30	Network Explorer	80
31	NAV v3 preferred installation layout	84
32	NAV v3 features	87



## Preface

It is Christmas time again, and time for another NAV report, or for some of us, time for the *making* of another NAV report. This report ends the fifth year of NAV development; it also ends the fourth NAV project. We have already had NAVMe, NAVRun and NAVMore. Now it is tigaNAV (and for our ignorant none-Indonesian speaking audience we should immediately explain... tiga means 3...).

tigaNAV is by far the biggest project – both in hours and personnel. More than 3500 work hours are spent, a total of 14 developers have been involved. An impressive amount of code is produced. The NAV-developers mailing list has seen a total of 1176<sup>3</sup> postings! There have been many discussions, many ideas, loads of enthusiasm and pure good will.

And – let there be no doubt – there has been quality work. This 92 page report is packed with results. Say no more.

The project leader would like to thank the entire development team. You should be proud – every one of you! After all, you have made NAV v3 possible. No more – no less.

And please – hang in there. Another year is coming up. And more ideas, I'm sure (yes, I know, Morten, requirements spec first ;)).

Last but not least, I would like to thank ITEA and UNINETT for believing in us, for giving the tigaNAV project the necessary resources.

Vidar Faltinsen

---

<sup>3</sup> This is actually 33% of all postings that have been on the list since the very first on June 30, 1999 (yes, I have them all).

# 1. Introduction

## 1.1 What initiated this project

Project tigaNAV is the fourth NAV collaboration project between NTNU and UNINETT. The project has been running from June to November 2003. The focus has been on further development of NAV version 3.

Currently the latest available version of NAV is version 2.1.5<sup>4</sup>. Development of NAV version 3 started already in 2002 with project NAVMore. Some of the NAVMore results (the service monitor in particular) have been in alpha production at NTNU throughout 2003.

Project tigaNAV has continued in the footsteps of NAVMore. The ultimate goal has been to make available a stable version 3 of NAV for NTNU and for UNINETT members.<sup>5</sup>

## 1.2 The Project Objective

tigaNAV introduces improvements on many aspects of NAV. The most important objectives of project tigaNAV have been:

- New and fundamentally improved NAVdb data collection system.
- Completely new GUI, more consistent look, use of templating.
- Flexible scheme for authorization allowing finer grained control of user rights.
- Message system to inform NAV users of planned outage, errors or other operational events.
- Device browser gathering all information on one device in one page with links to related statistics, reports etc.
- New web front end that replaces the text based seed files.
- A database (RRDdb) containing meta information on all statistical data. New mechanisms for sorting, structuring and combining statistics (RRD browser). Improved threshold monitor.
- A general module monitor that reports outage on modules in a chassis or a stacked (physically or virtually) switch.
- Device management with the ability to track milestone events of physical devices from order and arrival through the operational stages.

---

<sup>4</sup> As of now there is no publicly available licence of NAV. NAV is owned by NTNU. UNINETT has through project participation gained free access to the product, for itself and its members.

<sup>5</sup> For a complete overview of all NAV functionality see chapter 13.

## 1.3 Effects of the project results

NAV version 3 will ultimately be available to all UNINETT members. NAV v3 has many new features compared to NAV v2. It will even better aid the network management process. NAV v3 also takes its first small steps into the world of system management (service monitor, server statistics).

## 1.4 Personnel resources

A total of 14 persons have been involved in tigaNAV, some of them staff, many of them students. tigaNAV is a collaboration between ITEA's network group, ITEA's systems group, ITEA's user support group and UNINETT. The table below gives an overview:

Person	Organization	Focus
Vidar Faltinsen	ITEA network	Project leader
Morten Vold	ITEA network	Authentication, authorization, subversion repository, coordination of developers, installation/build
John Magne Bredal	ITEA network	Cricket, RRD, SNMP traps
Gro-Anita Vindheim	ITEA network	New front-end subsystems, emotD
Kristian Eide	ITEA network	gDD, OIDdb, cam logger, network and vlan topology discovery
Sigurd Gartmann	ITEA network	report generator, gDD, OIDdb
Magnus Nordseth	ITEA systems	Device browser, RRD
Stian Søiland	ITEA systems	Device browser
Arne Øslebø	UNINETT	Alert engine
Andreas Åkre Solberg	UNINETT	NAV profiles
Magnar Sveen	ITEA network	User interface
Hans Jørgen Hoel	ITEA support	editDB, device management
Erlend Mjåvatten	ITEA systems	RRD
Bjørn Ove Grøtan	ITEA systems	emotD

## 1.5 Subprojects and work hours

The project plan defined 12 subprojects. There has been activity on all subprojects. Most of the goals have been reached, some tasks have been postponed. We will comment on that in depth later in the subproject chapters of the report.

An overview of budgeted and used hours is shown. We also indicate objective achievement in percentage:

				Hours		Goal achievement
	Subproject	Ch	Participants	Budget	Used	
1	Authentication, authorization, profiles	2	Morten	160	155	90 %
2	User interface	3	Magnar, Morten	280	360	83 %
3	Event and alert system	4	Kristian, Arne, Andreas	280	200	90 %
4	Data collection system	5	Kristian, Sigurd	360	960	97 %
5	New front-end subsystems	6	Hans Jørgen, Sigurd, GA	260	345	70 %
6	RRD activities	7	John Magne, Erlend	440	375	75 %
7	Enhanced Message of the day	8	Bjørn Ove, Gro-Anita	120	80	75 %
8	Device browser	9	Stian, Magnus	400	430	75 %
9	Round trip and packet loss	10	Magnus	40	20	50 %
10	New network utilities	11	Sigurd, Gro-Anita	80	40	30 %
11	Version control and software build	12	Morten	80	120	80 %
	Installation, alpha test		Morten	0	110	
12	Project administration		All	240	350	
	<b>Total</b>			<b>2740</b>	<b>3545</b>	

As the table indicates we have used more time than estimated. Most of the subprojects have been roughly correctly estimated, there is one exception however; subproject 4.

In subproject 4 we have done a complete rewrite of the NAVdb collection system. There are many advantages with the new system, which we elaborate in chapter 5. We have to admit though, that we underestimated the task at hand. And since a consistent NAVdb is so fundamental to the rest of NAV, delays in this subproject have been a problem. Fortunately we were running on the old collection system in the busy first months of the project.

The project plan had a scheduled deadline set for mid August. Most of the subprojects were finished within this deadline, but not all. The delays have been in subproject 1 (done in September) and subprojects 4, 5 and 7 (done in November).

Our initial plan was to have a running alpha installation at NTNU by October 1. Due to the mentioned delays this has been postponed. At the time of writing this alpha test is finally being established. We elaborate on future release plans in chapter 13.2.

## 1.6 Outline of this report

Chapter 2-12 of the report describe in detail the results of all subprojects (1-11). Each chapter follows the same outline. The length may vary, typically in correspondence with the number of hours spent on the subproject.<sup>6</sup>

Chapter 13 gives an overview of all NAV v3 features and a suggested plan for further development and release.

Chapter 14 gives a summary of the project work and concludes this report.

## 1.7 NAV glossary

The following NAV specific terms may be used in this document:

<i>Term</i>	<i>Meaning</i>
Device	Physical device identified by a unique serial number.
Netbox / IP device	IP device; i.e. a device that is configured with an IP address.
gDD	getDeviceData – the data collection system.
vlanPlot	The network load map of NAV v2
NAVdb	The database containing the network model, the heart of NAV.
RRD	Round Robin Database, open source solution.
RRDdb	A part of NAVdb. Stores meta information on all RRD statistics.
OIDdb	A part of NAVdb. Contains information on the SNMP data we poll from the equipment.
editDB	The NAV v3 “Edit database” tool that replaces the NAV v2 seed files.
eMOTd	The NAV v3 message system

---

<sup>6</sup> This report is a collaboration effort with contributions from many project members. Thus the style of writing may vary from chapter to chapter.

## 2. Authentication, authorization, profiles

### 2.1 Resources

Subproject number	1
Subproject leader	Morten Vold
Developers	Morten Vold, Magnar Sveen
Hours	Budget: 160 Used: 155
Objective achieved	90%

### 2.2 Main objective

Whereas NAV v2 was dependant upon an external user database and authentication scheme (provided by Apache and flat password files), we want NAV v3 to have an integrated database of user accounts. We want authentication during web login to be integrated in the user interface, and login sessions to be managed with cookies. Also, we want a more fine-grained control over user authorization, through implementing the concept of an unlimited number of account groups. These objectives also spawn the need for an account (user) administration panel.

### 2.3 Results

#### 2.3.1 User database

The user database was adapted from the work done by Andreas Åkre Solberg (UNINETT) on the Alert Profiles system (see 4.3.4), and is contained within the "NAV profile" database (see figure 6, chapter 4). The database stores accounts, account groups, privilege information for account groups and preference settings for accounts. According to the project plan, the user database is feature complete - although testing of external user insertion has not been performed (such as importing from NTNU's BDB system).

#### 2.3.2 Authentication and authorization

A python module (using mod\_python for Apache) has been written to intercept all requests to the Apache server running NAV. This module performs authentication and authorization of every request. If either authentication or authorization fails, the web client is redirected to the NAV login page. If both are successful, control of the request is returned to Apache, which decides what to do with it. This means that any document on the NAV web server can be protected by NAVs authorization scheme.

Authentication is password based. Passwords are stored as MD5 hashes in the user account database. The authentication method should be transparently replaceable.

An authorization scheme has been established. Privileges may be granted to account groups based on a "subject, action, target" (or "who, what, where", if you will) principle. Whenever a piece of code needs to establish whether the logged in user has sufficient privileges to perform a specific action, it asks through a single API call "Is user X allowed to do Y with Z?" Privileges cannot be granted to individual user accounts, only to account groups. The privileges of an individual user are the combined privileges granted for the groups the user is a member of.

At the time, the only privilege name in use is "web\_access". The `mod_python` module that performs authorization uses this privilege name to establish whether the authenticated user is allowed to retrieve the URLs he/she is asking for in a request. The same functionality is used to limit which hyperlinks are displayed to an authenticated user in several subsystems (why display hyperlinks to URLs the user is not allowed to fetch?). Translated more literally into the "subject, action, target"-scheme of things, the module calls the API function to ask: "Does user X have web\_access privilege to `/some/document/url?`".

All logic to determine the outcome of such privilege questions is contained within one API call. As of today, privilege specifications are stored in the "NAV profiles" database, but they could just as easily be stored in a different system, on a different server. Only the one API call needs to be changed for all of NAV to make this possible. Many of these ideas come from thoughts of integrating NAV with external user authentication systems, such as the FEIDE project of UNINETT.

#### **2.3.2.1 Ideas for a typical Authorization setup**

Ideas on how to apply the flexibility of the new authorization system to a NAV installation at NTNU (and to a default NAV installation) have been discussed. The basic scheme is to grant the most typical combinations of privileges to a small set of account groups. Whenever a new user is added to NAV, his/her access privileges are set by adding him/her to a select subset of these groups.

The proposed default groups are as follows (group names may vary from final spec.):

- Anonymous\* Any unauthenticated (not logged in) user. The privileges granted to this group will be the minimum set of privileges for any user, and should therefore be set at a minimum (all users are considered members of this group during authorization of any kind).
- ReadMyOrg This group is typically intended for faculty IT personnel, but anyone that needs to be granted access to view network information within their organization can also fit this group. At NTNU, probably anyone who receives login access to NAV will be a member of this group.
- Privileges are granted to the all of the web tools, except editDB (see chapter 6.3.1) and NAV/User administration tools.
- Privileges are granted to view/receive information on any IP address within the prefix ranges of the organizations the user belongs to, and to view netboxes that belong to any of the organizations the user belongs to.
- Privileges should also be granted to see/receive information about any router or switch (The idea is that everyone should be able to see their way out through the network).
- ReadAll This group is aimed at users that should be able to see information about everything on the network.
- Privileges are granted to the all of the web tools, except EditDB and NAV/User administration tools.
- Privileges are granted to view/receive information on any IP address.
- Privileges should also be granted to see/receive information about any router or switch.
- Nett This group is intended for the network maintenance crew (may be applicable only to NTNU)
- Privileges are granted to the crew's repository of maintenance documentation and to local NAV additions created by the crew themselves.
- Sdrift This group is intended for the server/service maintenance crew (may be applicable only to NTNU).
- Privileges are similar to the Nett group.
- Admins\* This group is automatically granted all privileges.

---

\*This group is a built-in system group in NAV and will always be present.



- |            |                                                                                                                                                                                                                                                                                                     |
|------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| SMS        | This group is granted the privilege of receiving SMS messages in Alert Profiles.                                                                                                                                                                                                                    |
| WriteMyOrg | <p>This group is intended for faculty IT personnel that need privileges to add or change netboxes within their faculty.</p> <p>Privileges are similar to the ReadMyOrg group, but using <i>boxwrite</i> privileges instead of <i>boxread</i>, and also providing access to the editDB web tool.</p> |

The proposed set of privileges that can be granted to these groups are as follows:

- |                     |                                                                                                                                                                                                                                                                                                                                                                                                                          |
|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>web_access</i>   | Access to a specific URL. The target is stored as a regular expression – every request to authorize web access to a specific URL will attempt to match this URL against these regular expressions.                                                                                                                                                                                                                       |
| <i>boxread</i>      | Access to see/receive information about a specific netbox or IP address. The target is stored as an expression; examples are “ip in myorg”, “ip in 129.241.75.0/24”, “owner = myorg”, “cat in gw,sw”. When asked to check whether a user has privileges to view information on a particular IP address, the API call will evaluate these expressions on the fly, extracting information from the netbox table if needed. |
| <i>alerttype</i>    | Access to receive alerts through a specific service, such as SMS or Instant Messaging. Target is simply a mnemonic for the service in question. The group SMS (as proposed above) would be granted the alerttype privilege to the target “sms”.                                                                                                                                                                          |
| <i>reportaccess</i> | Access to specific reports in the report generator. The target can be stored as a single name or list of names of specific reports. This privilege will be used by the report generator alone.                                                                                                                                                                                                                           |
| <i>boxwrite</i>     | Write access using the same target semantics as the boxread privilege.                                                                                                                                                                                                                                                                                                                                                   |

### 2.3.3 User admin panel

A user account administration panel has been implemented. It allows for creating/editing/deleting accounts and account groups, and has a rudimentary interface for granting and revoking privileges to account groups.

It also has support for associating user accounts with organizations. Organizational memberships will later be used by the authorization system to determine which network information a user is allowed to see.

### 2.3.4 Sessions

Session handling is in place, using cookies for session identification. Any value can be made persistent during the lifespan of a session. Session data is stored as a "pickled" (serialized) Python object in the file system.

When a web client requests a document on the Apache web server, the authentication/authorization module takes control and tries to determine whether the request belongs to an existing session. If not, a new session object is generated with a unique 32-letter identification string. The id string is posted as an HTTP cookie to the client. In future requests, the web client will return the cookie containing the id string to the server, and the authentication/authorization module will load an existing session object identified by this string.

The cookie will only exist for as long as the user's web browser is open. Next time the browser is opened and requests a document on the NAV web server, it will no longer submit the cookie, and a new session object will be generated. In the file system we now have a dead session file, which should be removed. It is also good security practice to expire sessions that have not been in use for a number of minutes. Regular deletion of dead/expired session files has not yet been implemented, and old files will clutter the server's temporary directory until deleted manually.

## 2.4 Problems

A few mysterious problems with the login system plagued the developers during the project, but they were eventually solved.

### 2.4.1 Double login

One of these problems was having to log in twice before one was granted access to a page. The problem turned out to be the cookie mechanism. Developers working from within the itea.ntnu.no domain could reach the development server as *http://isbre/*, whereas the fully qualified domain name of the server was *http://isbre.itea.ntnu.no/*.

When retrieving *http://isbre/* and logging in there, the server would return a new session cookie, which the browser registered as belonging to the domain name *isbre*. Upon login, many pages would redirect the client to the fully qualified domain name address of the server. The browser would not recognize *isbre.itea.ntnu.no* as the same server as *isbre*, and would therefore not return the cookie set by *isbre*. Not receiving a session id cookie with the request, the server would promptly generate a new session object and send a

new cookie. As a result, the login credentials used with the previous session were lost, and the login page would display again.

Developers who consistently used *isbre.itea.ntnu.no* as their address for the development server would never see this problem. The problem was solved by configuring Apache itself to redirect any requests made against an unqualified hostname (such as an IP address or the abbreviated *isbre*) to the fully qualified hostname.

#### 2.4.2 Strange session related error messages

An attempt to read a session file that was simultaneously being written to by another web process would yield strange error messages that seemed random to the viewer. The solution was to implement locking of the session file. An exclusive write lock is placed on the file before it is written, and a shared read lock is placed on the file before it is read.

### 2.5 Further work

Not all objectives were achieved in detail. These things need to be further worked on:

- The user admin panel lacks a proper design template. A rough template is in place; this should be modified to make the interface more pleasant-looking.
- The session handling lacks automatic cleanup of dead and/or expired sessions. A cleanup function exists, but a system for calling it regularly must be devised.
- The only system that actually uses the authorization scheme so far is the URL-checking python module. The authorization API call must be extended to provide functionality for authorizing different kinds of actions against IP address ranges and netboxes, etc., based on the organizational memberships of users. These ideas have been described previously, but implementation has been postponed because more important features have been prioritized.
- A script must be written to perform the Python API privilege-checking call from the command line. This script can be used by non-Python scripts and programs to interface with the privilege system of NAV, thus the functionality need not be duplicated for every programming language.

### 2.6 Concluding remarks

Subproject 1 has successfully accomplished most of its objectives. The missing features listed under "Further work" must be implemented before the official release of NAV 3.0. Although the entire tigaNAV project has been belated, subproject 1 has come very close to its budgeted hours.

## 3. The User Interface

### 3.1 Resources

Subproject number	2
Subproject leader	Morten Vold
Developers	Magnar Sveen, Morten Vold
Hours	Budget: 280 Used: 360
Objective achieved	5/6 (83%)

### 3.2 Main Objective

The main objectives of this subproject are:

1. The web interface in all NAV subsystems are implemented in the same language; Python. (one exception: alert profiles)
2. A common scheme for templating is used. Within a template html and styles are separated using CSS. (complete)
3. tigaNAV introduces a new NAV design, possibly with scroll down menus, new side bar, if any. (complete)
4. There are mechanisms for the template system to easily integrate the NAV environment within each subsystem. (complete)
5. A general concept/file structure for all subsystems. Each subsystem gives structured meta information about itself (icon, description, url etc). (complete)
6. User profiles may alter the look and give a personal view. (not complete)

### 3.3 Results

#### 3.3.1 Templating solution

The Python templating engine Cheetah (<http://www.cheetahtemplate.org/>) was chosen in order to help separate content, graphic design and program code.

See figure 1 for an illustration of how Cheetah works. During development a .tmpl file is created. This file looks like html-code, but includes Cheetah flow control and variables. The template file is compiled to a run able .py file when first installing the product. When the web server gets a request for a page using the template, the .py template is used to create an html-formatted file that is sent to the client.

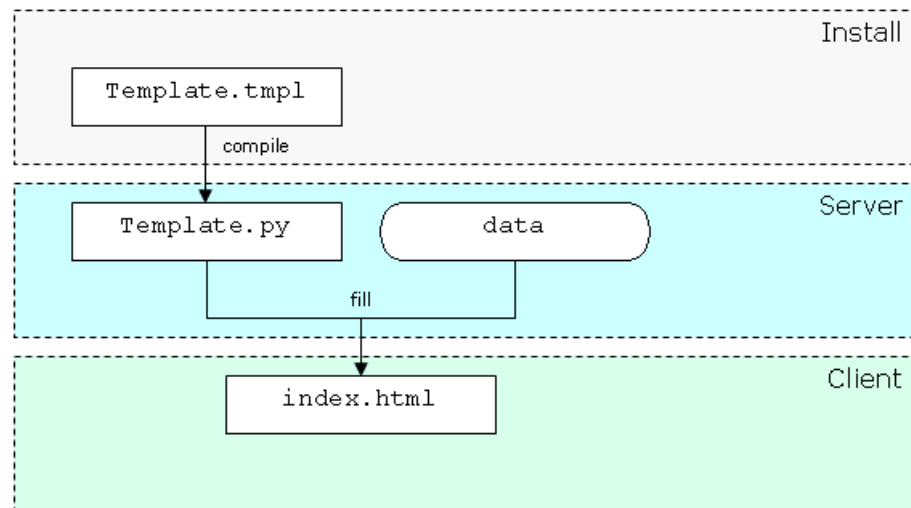


Figure 1: How Cheetah works

For NAV the most notable template is `nav.web.templates.MainTemplate`, containing the NAV web interface. All other templates inherit from `MainTemplate`.

Some templates were developed specifically for a tool, allowing the programmer to focus on delivering data, and not on looks and web design. The tools in question were the Report and Machine tracker tools.

All other NAV tools use the main template, easily inserting their pages by replacing the template's *content*-function.

### 3.3.2 Web interface

The focus of the NAV web interface is on quick navigation and easy access to the tools you use the most. Take a look at figure 2 for an overview of the different elements making up the standard interface.

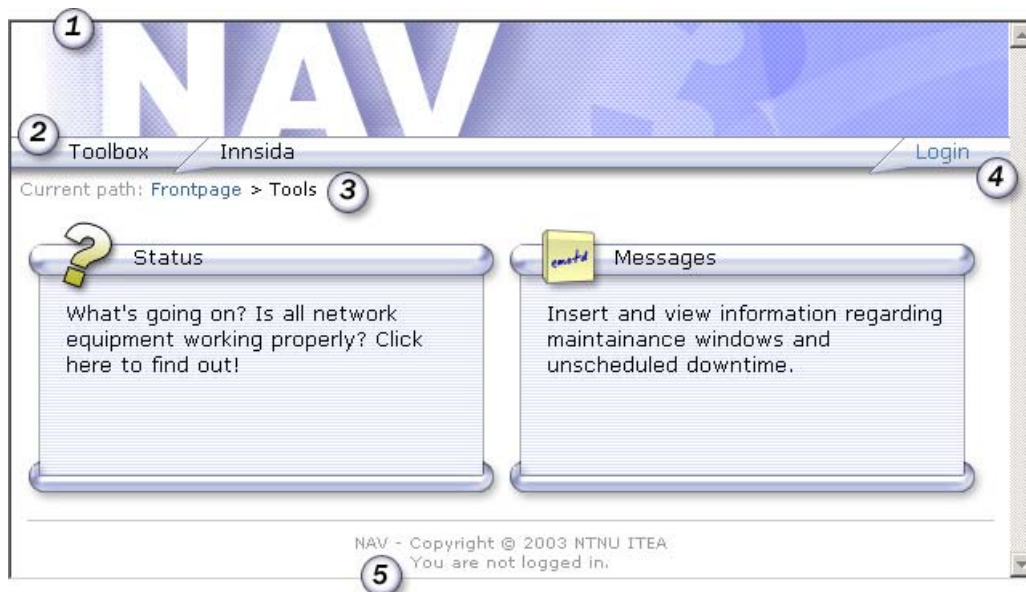


Figure 2: The web interface

1. NAV logo. Clicking the logo will always lead back to the front page.
2. Quick link bar. Individually customizable, this is the best place for links to frequently used tools and web pages.
3. Navigation info. This shows where in NAV you are at any time, giving an overview of structure and easing navigation of the page.
4. Login/logout button. To get access to restricted information and use your own preferences, log in here.
5. Footer. Copyright notice and information about what user is logged in.

The content area is on a simple white background. There are a lot of tools in NAV, and more are being developed. Most of these focus on function and not form. With a neutral white background, the design of these tools does not look out of place.

### 3.3.3 Front page

The front page serves two main purposes. For the new user, it welcomes and describes the purpose of the site, and presents contact information. Other users are interested in the current status, planned downtime etc. This is presented as Messages of the Day and Status Now. See Figure 2.3.

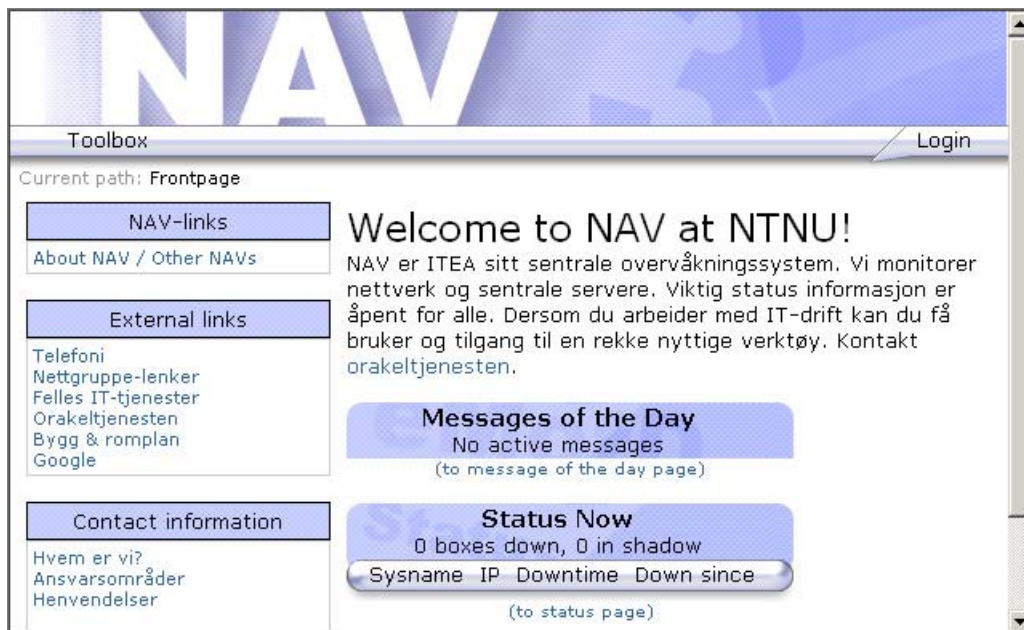


Figure 3: A front page with no eMotD or boxes down

The information on the front page is limited to the most recent messages and dead boxes. More information and more entries are available from their respective pages.

There are also internal NAV-links and external links. The internal links are checked towards the user's authorization before they are shown. All static information on the front page is easily customizable.

### 3.3.4 Toolbox

The toolbox is an easily accessed collection of tools, mainly aimed at the new user of NAV. Each tool features a unique icon and is verbosely described.

Metadata about the tools is collected and maintained in the same location, making it easy to add, remove or change information about tools at a later date. The idea is that new tools can be developed and released to all NAV installations with ease.

All tools the user doesn't have access to are filtered out. A new user isn't flooded with tools he or she isn't allowed to use - and probably isn't interested in using. The users can easily identify the tools available to them. The toolbox for an anonymous user on the development box for NAV v3 can be seen in figure 2.2.

Senior users might feel the toolbox is cluttered because of the sheer number of tools - but these users aren't the intended users of the toolbox. By using

the navigation bar preferences menu, they can easily add quick links to the tools they use often.

### 3.3.5 Quick link preferences

The quick link bar (see 2 in figure 2) is fully customizable for a registered user. A user can choose from a set of default choices, or add his or her own links.

This is the easiest way for a user to get quick access to frequently used tools and other resources. The quick link bar stays the same, independent of what page in NAV the user is browsing.

Figure 4: The quick link preferences page

See Figure 2.4. The leftmost box represents the navigational bar, and what links should appear there. In addition there are two optional quick link boxes that represent dropdown menus, to allow for large amounts of links. These can be turned on and off using the checkboxes on the top.

Personal links can be removed, edited or added by clicking the trashcan icon, pen-and-paper icon or the *Add personal link* button, respectively. This is the same interface that a site admin uses to edit the default links for anonymous and new users.



### 3.4 Problems

The site was designed using Cascading Style Sheets version 2. Some older browsers do not support this very well. While the site looks good in all new browsers, some old browsers will look strange.

Forms for filling in information need to be hand written since they are very rarely equal in form and function. Making all forms have the same look requires some work - therefore it hasn't been a priority.

In order to set the defaults for anonymous and new users in the quick link preferences, you need to be logged in as site admin. Giving this functionality to other users isn't supported.

### 3.5 Further Work

In addition to fixing the problems mentioned in the previous section:

- Unique icons for all subsystems should be created. As of now, some are blank while others are re-used.
- Only a few of the subsystems have been given special attention for design work. Programmers of some other subsystems have requested a design overhaul.
- While all subsystems have icons, they're at the moment only used for the toolbox. Displaying these icons while using the actual tools would be worthwhile, if an unobtrusive spot was found.
- The quick link preferences page (Figure 2.4) could use some work to make it more intuitive.
- The quick link dropdown menus are named "Quick link #1" and "Quick link #2". Some easy way to change the name of your personal dropdown menu would be a nice feature.

## 4. Event and Alert system

### 4.1 Resources

Subproject number	3
Subproject leader	Vidar Faltinsen
Developers	Kristian Eide, Andreas Åkre Solberg, Arne Øslebø
Hours	Budget: 280 Used: 200
Objective achieved	90%

### 4.2 Main objective

- Support for new event types in event engine, i.e. module State and threshold State (completed).
- SMS support with Alert Engine (completed)
- Support for new filter matches (achieved). Improvements in the NAV profiles GUI, make it more intuitive, less complex (partly achieved). Make NAV profiles a more integrated part of NAV (the goals we aimed for completed)
- Jabber support (postponed)

### 4.3 Results

The Event and Alert system was developed in project NAVMore. See the NAVMore final report for details<sup>7</sup>. The system is “fresh”, with tigaNAV we saw the need to elaborate on some important aspects. A recap of the overall picture is given on figure 5.

#### 4.3.1 Event Engine

Event Engine processes events generated by other systems in NAV; it is plug-in-based and the plug-ins do the actual processing of incoming events. The plug-in will take any necessary action and decide if sending an alert to Alert Engine is warranted.

New in event Engine is the addition of two plug-ins:

- MaintenanceState  
Netboxes (and services) can be put on maintenance; when in this state the actual sending of alerts related to the netbox (service) is disabled. All events are processed however, and will thus appear in the alert history, and status for the netbox, as seen on the NAV web page, will be updated

---

<sup>7</sup> The NAVMore report (in Norwegian): <http://metanav.ntnu.no/NAVMore/NAVMore.pdf>

as normal.

- ThresholdState

NAV monitors several statistics on devices, and when a predefined threshold is exceeded a threshold event is generated; when the statistic later falls below another predefined limit another event is generated. This plug-in does text formatting and sends an alert to alert engine.

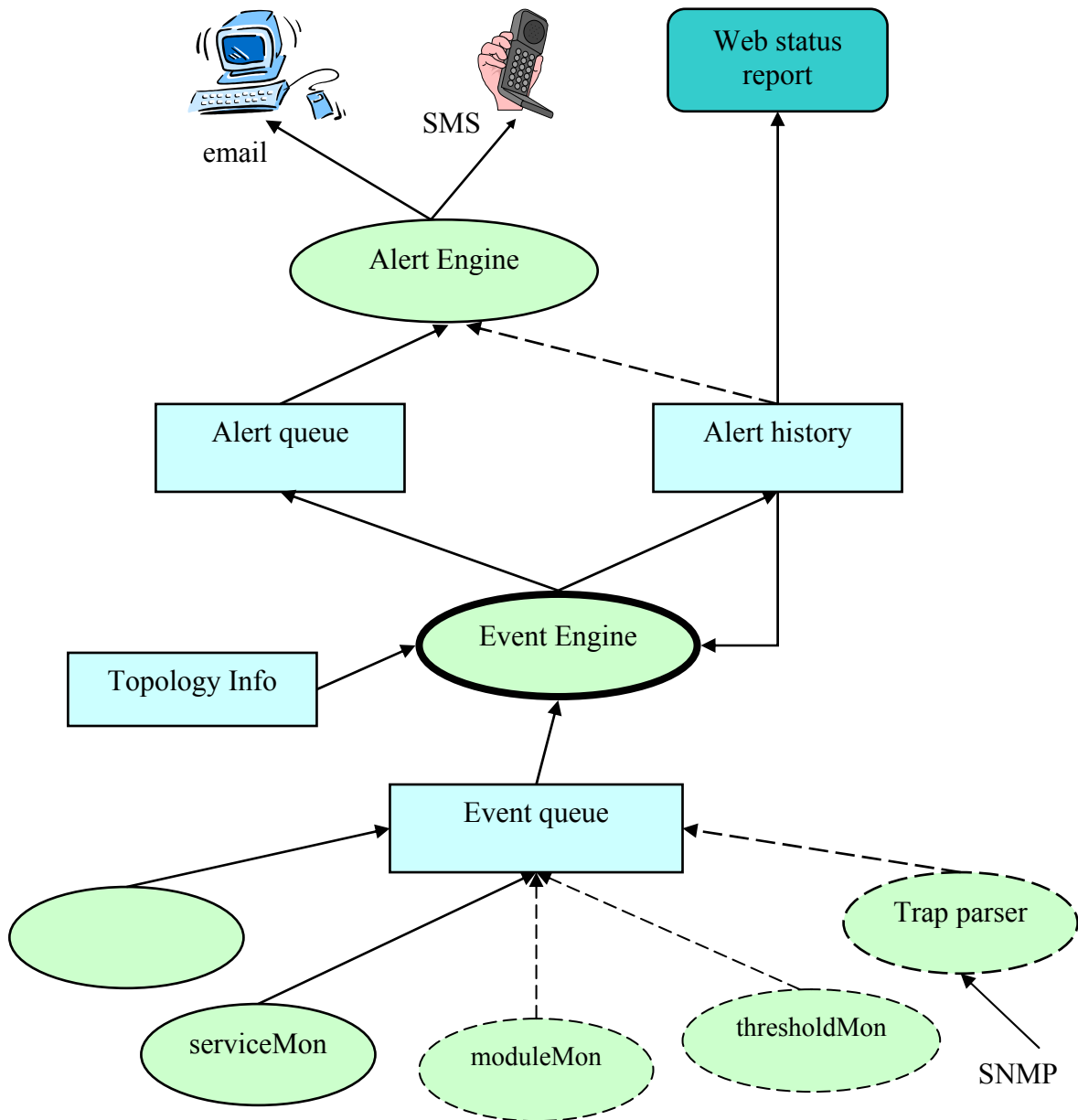


Figure 5: The Event and Alert System

A note on the 'severity': each event and alert has a given severity-value, which is an integer defined to be between 0 and 100, where 100 means the highest severity / priority. For example, users can configure their alert profile to only send alerts during the night for alerts with a severity higher than a specific value.

Currently, boxShadow and boxSunny (events sent instead of boxDown and boxUp, respectively, when a netbox appears to be unreachable because of another netbox being down) alerts will have their severity lowered.

#### 4.3.2 Alert Engine

We have not done much work on Alert Engine in tigaNAV, only minor improvements and bug fixing.

#### 4.3.3 The SMS daemon

The SMS daemon is ported from NAV v2 to NAV v3. No major changes have been implemented, only adjustments to the new environment with Alert Engine. The solution has been tested and works fine.

#### 4.3.4 Alert Profiles

Development of Alert Profiles started in the early summer of 2002, first as an independent UNINETT project, later with the ambition to deliver an integrated NAV solution through the NAVMore project. The focus in tigaNAV has been to continue this integration process. We have worked with:

- **Integrated web look**  
Alert Profiles is written in PHP (and will remain that way), while the new NAV GUI (see chapter 3) is based on a python templating solution. The challenge has been to transparently include the dynamic NAV header with personal user information etc. Our solution has been to implement a python script that takes a username as input and returns an empty template. This script is in turn called from Alert Profiles.
- **Integrated user database**  
Alert Profiles has a separate database with information on all users and their alert profiles (a sensible solution, also analogous to the NAV v2 trapdetect database). In tigaNAV we have integrated the work done on authentication and authorization into the Alert Profiles database. Figure 6 gives an updated picture. The tables 1-4 are related to the user accounts. All management of these tables is now outside the Alert Profiles GUI, and instead done in the user account GUI.

- **User documentation**

Extensive user documentation is written (in English). There are three main sections: simple usage, advanced usage and administration. A PDF is available from <http://pot.uninett.no/~andrs/nav/>

## NAVprofile

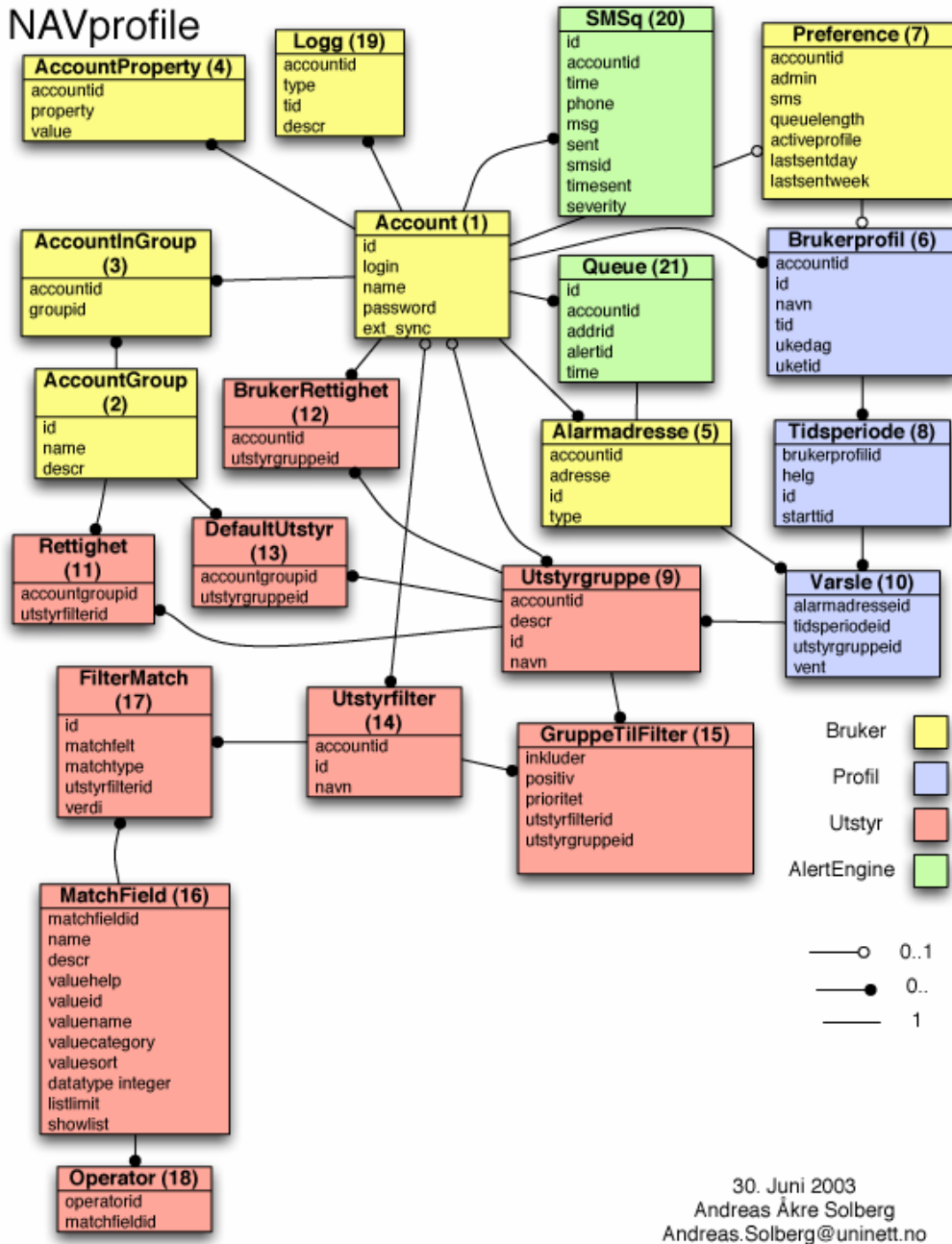


Figure 6: The NAV Profiles database

- **More extensive support to make filtermatches.**

NAV users set up their alert profiles based on equipment groups, which in turn are based on filters. Filters are made up of filtermatches with matchfields. Prior to tigaNAV the matchfields were hard coded definitions, shared among the Alert Profiles web interface and the Alert Engine backend.

Instead of hard coded values, the filtermatches now contain a relation to a row in the matchfield table. The matchfield table, contains a set of predefined matchfields. A matchfield is directly related to the tables and fields in NAVdb. In addition matchfields contain information on how to sort, group and present the matchfields to the users. Matchfields also contain optional relations to an operator table, determining which operators to use with the matchfield when creating filtermatches. The relation between filtermatches, matchfields and operators is illustrated in table 14, 16, 17 and 18 of the NAVProfile database (see figure 6).

A new tool for administrating matchfields is available for Alert Profiles administrators. Dynamic dropdown menus can be used to create relations to NAVdb

- **More intuitive GUI**

Alert Profiles is very feature rich with many possibilities. The focus has been on maximum flexibility for the NAV user, the downside being complexity. We have to focus on ease of use. Some measures have been taken in tigaNAV, more will come. The most important is the new Alert Profiles main page which immediately gives an overview of the user's active profile. An example is given in figure 7 below. The example user has chosen SMS alarms for all network devices during work hours and a more limited profile for weekday evenings and nights, yet another profile for weekends.

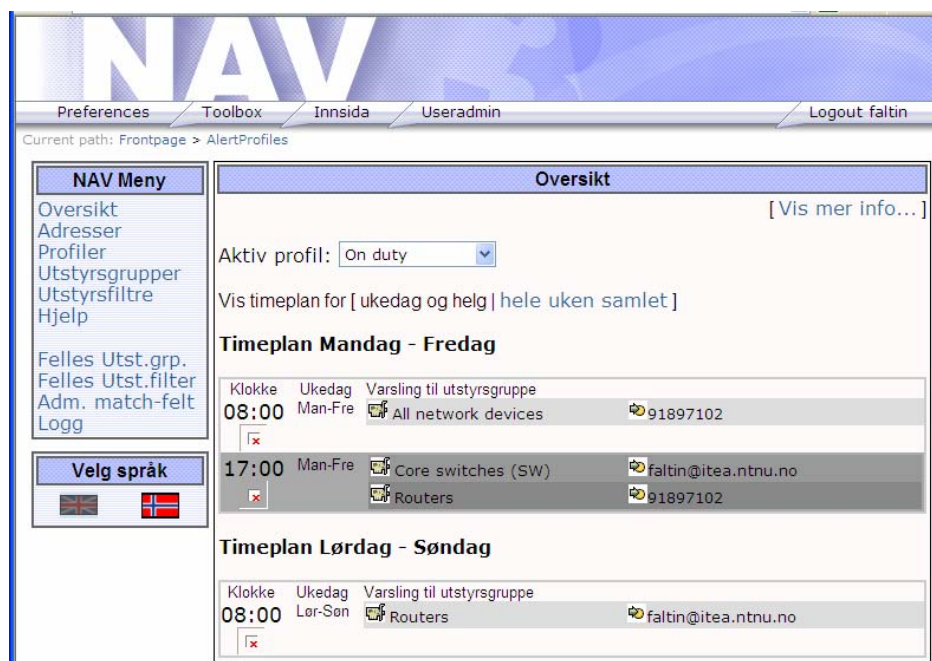


Figure 7: The Alert Profile main page

## 4.4 Problems

Some problems occurred when Alert Profiles was moved to the new development machine `isbre.itea.ntnu.no`. Problems were related to the multilingual support. Multilingual support depends on the GNU `gettext-tool`. `gettext` is not properly setup on `isbre`. The Alert Profiles web interface still works, but only in Norwegian.

## 4.5 Further work

### 4.5.1 Event Engine

- More fine-tuning of `BoxState` and `ServiceState` plugins to Event Engine; this will reduce response time even further while disregarding false alarms.
- Better control over severity values; this will make it easier to filter out uninteresting events and giving more attention to important events using the alert profile.
- More use of the events system for communication between NAV systems; using the event queue to notify other systems will allow said systems to update themselves immediately to changes, for example the addition of a new netbox.

### 4.5.2 NAV Profiles

- Displaying the definition of equipment groups more user friendly
- Converting the GUI and the database to English (part of the database is translated).
- Use `hasPrivilege` to decide a users admin privileges and if the user has permission to receive SMS messages.

## 5: The NAV v3 data collection system

### 5.1 Resources

Subproject number	4
Subproject leader	Gro-Anita Vindheim
Developers	Kristian Eide, Sigurd Gartmann
Hours	Budget: 360 Used: 960
Objective achieved	97%

### 5.2 Main Objective

- NAVdb design adjustments to achieve an even better model of the running network (completed).
- Replace the existing collection system with getDeviceData (completed, 60% of the subproject work hours)
- Introduce an OID database for a more flexible classification of equipment types and adhering OIDs (completed, 20% of the subproject work hours)
- Implement a general module monitor that sends alarms on outage of modules in switch stack (of any vendor) (completed).
- Elaborate on the collection of inventory data from routers and switches (postponed).
- Update the cam logger, the network topology discovery and the vlan discovery to the new environment (completed, 15% of the subproject work hours)

### 5.3 Data collection in NAV v3 at a glance

<i>Item</i>	<i>NAV v3</i>	<i>NAV v2</i>
Central OID database	Yes	No
Easy adding of new types without involvement of NAV developers	Yes	Partial
Plugin-based architecture for data collecting	Yes	No
Fine-grained control of collection intervals	Yes	No
Support for VLAN reuse	Yes	Partial
Detailed information for both switch and router modules	Yes	No
Store arbitrary information about a netbox	Yes	No



## 5.4 Subsystem overview

### 5.4.1 NAVdb updates

The NAV central database, the *NAVdb*, has been significantly updated to more closely model real networks. With this more detailed and realistic world view we can store more information and give richer reports. Major enhancements include easier maintenance, full support for VLAN reuse and accurate modeling of multi-module switches and routers.

### 5.4.2 getDeviceData

A major new component of NAV v3 is *getDeviceData*, the central subsystem handling all aspects of data collection from network devices (with the exception of Cricket data). In NAV v2 this was handled by a loose collection of scripts running every night.; problems included slow updates, no control over what was collected (all-or-nothing) and poor code reuse, complicating updates.

*getDeviceData* is a continuously running program based on *plugins* and the new *OID database* (see next section). This allows a much higher degree of flexibility on collection scheduling and very easy support for new device types; modularity and avoidance of code duplication greatly reduces the time spent on maintenance.

### 5.4.3 The OID database

All OID information is now centralized in the NAVdb instead of being hard coded in source code. This makes for easy overview of supported OIDs and also allows for fine-grained control over how often each OID is collected for each type of network device. It has also enabled us to automatically test new types for supported OIDs, thus significantly reducing the effort needed to support a new type of network device in NAV; in many cases this will be as simple as adding the type name and NAV will do the rest.

### 5.4.4 GetDeviceData Plugins

*getDeviceData* relies on *plugins* to do the actual work of collecting data and updating the NAVdb to reflect reality. There are two types of plugins: *device plugins* and *data plugins*; the former make use of the OID database to actually gather data from the network devices, while the latter is responsible for making the necessary changes to the NAVdb.

- Each data plugin presents a well-defined set of supported attributes which can be set by device plugins. Examples include the uptime of the device and the speed and duplex of each switch port on a switch.
- Each device plugin supports a set of OIDs, and after collecting one or more of these, depending on scheduling settings, talks to the relevant data plugin(s) to set the collected attributes.

The motivation for having two types of plugins is avoiding code duplication; two device plugins can collect the same attribute from two different types of devices, but updating of the NAVdb can still happen in a single location.

#### 5.4.5 The cam logger

The *cam logger*, responsible for the collection of MAC addresses and CDP data, has been updated to make use of the OID database. This has greatly simplified its internal structure as all devices are now treated in a uniform manner; the immediate benefit is that data collection is no longer dependent on type information and no updates should be necessary to support new types. Upgrades in the field can happen without the need for additional updates to the NAV software.

#### 5.4.6 Network topology discovery

A major feature of NAV is the ability to automatically discover the physical topology of the network based on collected MAC and CDP data. The network topology discovery system has been updated to the new NAVdb design, and received some minor enhancements and internal restructuring to facilitate maintenance.

#### 5.4.7 Vlan discovery

While the network topology discovery system maps the physical network, the vlan discovery system completes the picture by mapping the logical network, that is, the broadcast domains, or VLANs, as seen by the network users. Significant updates to fully support the reuse of VLAN numbers across broadcast domains have been integrated in the new version.

### 5.5 Detailed description of new subsystems

The following sections goes into detail on each of the new or updated subsystems involved in data collection for NAV v3, and significant changes from the previous version are noted. The most important goal of the new data collection system is, ironically, that users of NAV should be even less aware of its existence; it should “just work”. The most significant new subsystem is the *OID database* and its implementation in *getDeviceData*, the central data collector for NAV; most of the other changes, as we will see, follow naturally from the enhancements it brings.

#### 5.5.1 The NAVdb

The central NAV database, the *NAVdb*, contains a model of the world as seen by NAV. It has continually evolved with each new version of NAV to more accurately model real networks, and the latest version supports all common features of modern networks.

Figure 8 shows the most important tables; note that only fields significant to the new version of NAV are included.

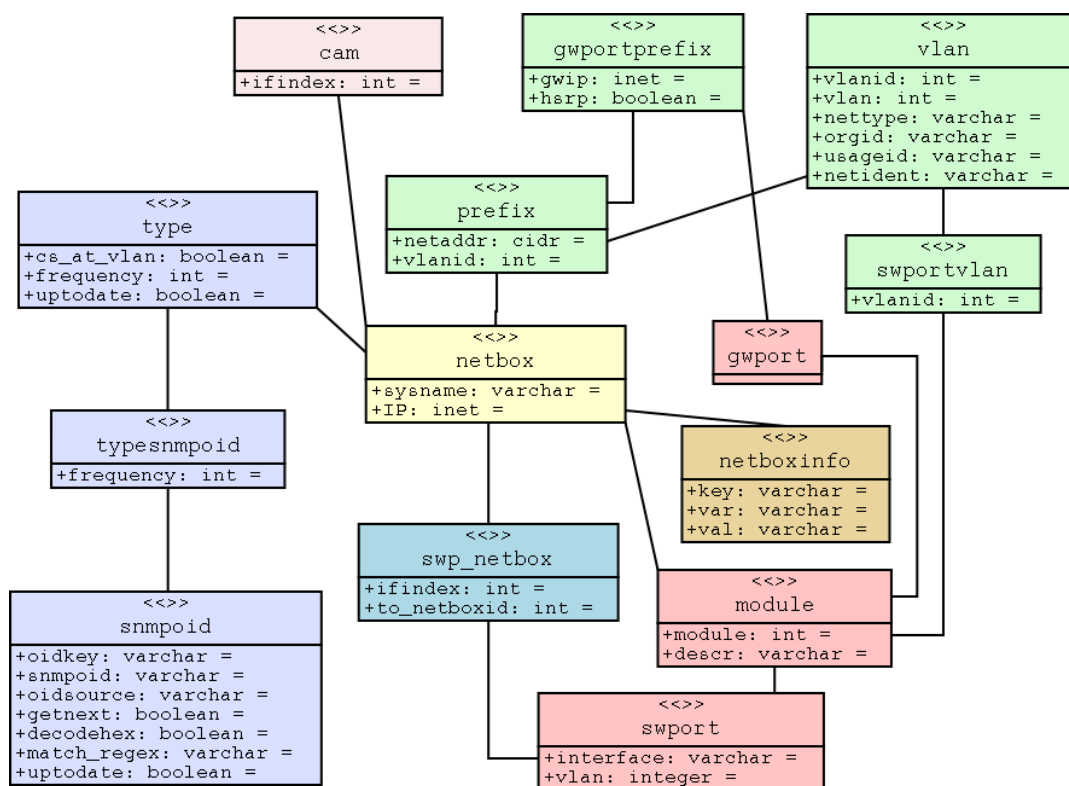


Figure 8: NAVdb with new and/or significant fields

Detailed description of each table and field:

## netbox

Describes a device with an IP address, and most also include SNMP capability. All other tables reference it either directly or indirectly.

type

Lists all types recognized by NAV. It comes with all common types already defined, and adding new types is easily accomplished with the new *editDB* web interface. In many cases this is all which is required to fully support a new type in NAV.

New fields in NAV v3 include:

- *cs\_at\_vlan* – type *boolean* – updated automatically  
Specifies if the type support appending @<vlan> to the community string to collect VLAN-specific information.
- *frequency* – type *integer* – optional  
Specifies the default interval time, in seconds, between data collection for the type if no time has been set for an OID.

- *uptodate* – type *boolean* – updated automatically  
Specifies if all OIDs have been tested against this type; used by the automatic OID tester, part of `getDeviceData`.

### **SNMPoid**

Lists all OIDs recognized by NAV. Each OID has a unique *oidkey* which identifies it; the *oidkey* is used instead of the numeric OID in the source code of programs. Fields:

- *oidkey* – type *varchar*  
Is unique and maps to a numeric OID (which does not necessarily have to be unique).
- *SNMPoid* – type *varchar*  
The numeric OID.
- *oidsouce* – type *varchar*  
Only used for descriptive purposes; name of vendor or standard which define the OID.
- *getnext* – type *boolean*  
Specifies if the OID is exact, that is, the GETNEXT SNMP type should not be sent in the first packet. It is important this field is set correctly to ensure reliable operation of the OID tester.
- *decodehex* – type *boolean*  
Specifies if the SNMP stack should try to interpret returned hex data as ASCII text.
- *match\_regex* – type *varchar*  
Used by the OID tester when determining if a type supports this OID; at least one response must match the given regex.  
*Note:* the whole response is matched against the regex, which should be designed to take this into consideration.
- *uptodate* – type *boolean* – updated automatically  
Serves the same purpose as *uptodate* for the *type* table. Specifies if all types have been tested against this OID; used by the automatic OID tester, part of `getDeviceData`.

### **typeSNMPoid**

Connects the *type* and *SNMPoid* tables; each type can support several OIDs, and each OID can be supported by more than one type. It also has one important additional field:

- *frequency* – type *integer*  
Specifies the interval time, in seconds, between data collection for the type and OID; overrides any default value for the type.  
*Note:* if the type has no default value and this field is not set, data for the given type and OID will **not** be collected!

### **module**

Lists all modules of both switches and routers; both *gwport* and *swport* now reference this table. Other changes include:

- module – type integer  
The definition has changed to *integer* from *varchar*. The full description of the module is stored in the *descr* field instead of a truncated version being used for this field. Counting generally starts at 0, and in the case of conflicting numbers the next available is used.
- descr – type varchar  
Stores the full textual description of the module for informational purposes.

#### **swport**

Lists all switch ports on each module of a switch. Generally unchanged except for the addition of two new fields:

- interface – type varchar  
The interface name. This should be the same interface name as seen by *CDP remote interface*, and is used by the *cam logger* to match the two.
- vlan – type integer  
Reintroduced from NAV v1 to support full VLAN reuse. It stores the VLAN number as set on the actual switch interface. The *vlan discovery system* uses this information to determine the real VLAN.

#### **swportvlan**

Lists the VLANs running on each switch port; for non-trunk ports this will only be one, while there can be several for trunk ports. The *vlan discovery system* is now solely responsible for updating this table and determines the real VLANs running on each port; it references the *vlan* table instead of storing the VLAN number directly.

#### **gwport**

Lists the router interfaces on each modules of a router. Only minor changes.

#### **prefix**

Lists all prefixes known to NAV. Almost all fields have been migrated to the new *vlan* table; only the *netaddr* field and a reference to *vlan* remain.

#### **gwportprefix**

Connects the *gwport* and *prefix* tables; multiple gwports can be on the same prefix, while each gwport can have IPs on more than one prefix. Also contains two addition fields:

- gwip – type inet  
The gateway IP.
- hsrp – type boolean  
Specifies if the gateway IP is a virtual IP as per the Cisco HSRP protocol.

### **vlan**

Lists all VLANs; all fields come from the old *prefix* table. The only change is that the uniqueness requirement of the *vlan* field is dropped.

### **swp\_netbox**

Describes all netboxes found behind each switch port. It is filled by the cam logger using MAC and CDP data; the topology discovery system uses the data for determining the physical topology between switches and their router uplinks. The only changes is that *ifindex* is now used to map to the swport instead of the module / port pair, which improve robustness.

### **cam**

All MAC addresses found on all switches monitored by NAV are logged to this table; very useful in abuse cases for tracking the origin of an IP. Changed from NAV v2 is that *ifindex* instead of the module / port pair is used to map to a swport. The module and port is still stored for archival purposes however.

### **netboxinfo**

General store for information about a netbox. Supports a two-level hierarchy using a key / var pair to map to one or more values.

### 5.5.2 getDeviceData

As mentioned in the overview section, `getDeviceData` is a major new component of NAV v3. It handles all aspects of data collection from network devices, with the exception of *Cricket* which still does its own data collection (also using the OID database). The following sections will go over each feature of `getDeviceData` in detail.

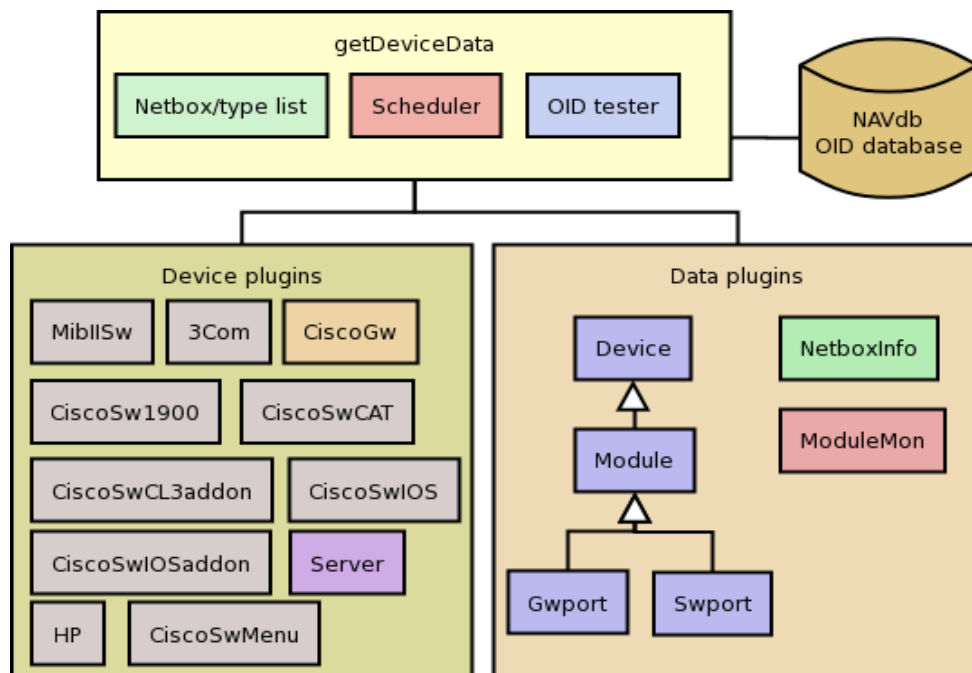


Figure 9: `getDeviceData` component overview

#### 5.5.2.1 `getDeviceData` core

The `getDeviceData` core provides supporting functionality for the plugins. This includes maintaining information about all netboxes and types, the OID database and scheduling data collection according to the intervals specified. `getDeviceData` is continuously running and thus has full control over all scheduling decisions.

#### 5.5.2.1 Netboxes and types

At startup, and later with a specific interval, fetches information from the NAVdb about all netboxes, their types and the supported OIDs for each type. The *frequency* field from *typeSNMPoid* is used to determine the update frequency for an OID; if not specified the default frequency for the type is used. If neither is set data will **not** be collected for this OID and type combination. The data fetched from the NAVdb is available to plugins when they are asked to do data collection.

### 5.5.2.3 Scheduling

`getDeviceData` internally uses two levels of priority queues to do scheduling; the first is the global queue containing all the netboxes; the second is one for each netbox containing its scheduled OIDs. The reason for having two levels is that different OIDs can have different intervals, and although there is no guarantee for when collection will take place (this is dependent on available worker threads) the interval must be strictly kept. That is, if one OID has an interval of 5 and a second an interval of 10, the OID with the 5 interval must always be collected together with the 10 as there may be dependencies between OIDs.

As most of the time during data collection is spent waiting for data from the network `getDeviceData` make use of worker threads to speed up the collecting. The number of threads is configurable and only limited by available computer resources; it is guaranteed that only one thread will request data from any one netbox at a given time, thus it is safe to use a high number of threads if allowed by machine resources and network bandwidth. The default number should be suitable for most NAV installations however.

### 5.5.2.4 The OID tester

The OID tester will try to collect OIDs for netboxes looking for a valid response; if one is received the type for the netbox is marked as supporting said OID.

The algorithm is simple, although parallel collecting makes it highly efficient even for larger number of OIDs and netboxes: Try to collect all OIDs from all netboxes. Once a valid response is received the type is said to support the OID and no further testing of this type and OID pair is necessary. For a response to be valid it must match the given regex, or in the case of the regex being empty any non-empty response is valid. Also, only one thread can collect data from the same netbox at any one time.

Once an OID has been tested against all netboxes or a valid response has been received the *uptodate* field of the *SNMPoid* table for the given OID is set to true. Also, when all OIDs have been tested against all netboxes of a given type, or a valid response has been received, the *uptodate* field of the *type* table for the relevant type is set to true.

The OID tester also tests for the *cs\_at\_vlan* property; this is done simply by trying to get the system OID with “@1” appended to the community string; if the device responds it has the property. An exception is made for 3Com switches which do *not* support this property, but still accepting the modified community string, ignoring the added postfix.

### 5.5.2.5 Plugins

`getDeviceData` supports two types of plugins which complement each other; having two types removes the need for code duplication and provides maximum flexibility. The two types are described in the following sections.



### 5.2.2.6 Data Plugins

It is the responsibility of the data plugins to update the NAVdb with collected data. Each data plugin presents a well-defined set of supported attributes which can be set by device plugins. Device plugins are free to make use of as many data plugins is necessary to store all collected data.

Note that some of the plugins are dependent on others to avoid code duplication. Refer to figure 9: *Module* makes use of *Device*, and both *Swport* and *Gwport* make use of *Module*.

Currently the following data plugins are available:

1. Device  
Updates the *device* table with attributes *serial*, *hwVer* and *swVer*.
2. Netbox  
Updates the *sysname* and *uptime* of a netbox.
3. Module  
Updates the *module* table with relevant fields; the *module number* is required.
4. Swport  
Updates the *swport* table with relevant fields; the *ifindex* is required.
5. Gwport  
Updates the *gwport*, *prefix*, *gwportprefix* and *vlan* tables. Since there is no available information to uniquely identify VLANs the updating process is somewhat more complicated than for swports.
6. NetboxInfo  
Allows storing arbitrary information about a netbox in a two-level hierarchy using a key / var pair to map to one or more values
7. ModuleMon  
The ModuleMon plugin is used for reporting modules not responding to event Engine, the central event reporting system in NAV v3. Currently the Midis device plugin will produce a list of responding ifindexes, and this module will check the list against known modules for a switch; any modules without responding ifindexes will be reported as down.

### 5.2.2.7 Device Plugins

The device plugins collect data from network devices and give it to data plugins for further processing using a well-defined API. Each device plugin supports a set of OIDs, and most importantly, how to interpret returned data

from devices and convert it into a standard format for processing by data plugins.

A device plugin is first given a netbox by `getDeviceData` and asked if supports collecting any data from it. The device plugin will then typically examine the intersection of the set of OIDs supported by the netbox and its own set of OIDs, and give a positive response if it is non-empty (or said in another way, if the plugin can collect at least one of the OIDs supported by the netbox).

After all device plugins have been asked, the ones which can support the netbox are asked to actually collect data; they are free to complement each other, that is, one plugin can collect the speed of a switch port while another collects the duplex. After all data collection is done the data plugins are asked to update the NAVdb to reflect any changes.

It should be noted that not all OIDs supported by both the netbox and the device plugin must be collected; it may be desirable to update the link status of a switch port more often than its speed for example. The implementation of device plugins should allow for this.

There are no dependencies between device plugins. Currently the following device plugins are available:

1. MibIIsw

Collects standard OIDs from the MIB-II standard. These include `ifSpeed`, `ifAdminStatus`, `ifOperStatus`, `ifDescr` and `ifName`. It will also report any modules not responding to the ModuleMon data plugin.

2. 3Com

Collects switch port state for PS40; speed, duplex and media type for SuperStack switches and also serial number, hardware version and software version for each module in a stack.

3. CiscoGw

Collects data about router interfaces from Cisco routers.

4. CiscoSw1900

Collects switch port data from C19xx switches; `c1900Duplex` and `c1900Portname`.

5. CiscoSwCAT

Collects the standard Cisco Catalyst switch port OIDs. These include: `portIfIndex`, `ifName`, `portDuplex`, `portVlan`, `portVlansAllowed`, `portTrunk` and `portPortName`.

6. CiscoSwCL3addon

Collects switch port data from Cisco CL3 switches. These include: ifTrunk.

7. CiscoSwIOS

Collects switch port data from Cisco IOS switches. Includes ifDescr, ifName, ifVlan, ifVlansAllowed and portPortName.

8. CiscoSwIOSaddon

Collects extra data from Cisco IOS switches. Includes iosTrunk and iosDuplex.

9. CiscoSwMenu

Collects switch port data for C3000/C3100 series. Includes cMenuIfIndex, cMenuPortStatus, cMenuPortType, cMenuDuplex, cMenuTrunk and cMenuVlan.

10. HP

Collects switch port data from HP switches. Includes hpSerial, hpHwVer, hpSwVer, hpPortType and hpVlan.

11. Server

Collects data from servers.

### 5.5.3 The cam logger

The cam logger collects the bridge tables of all switches, saving the MAC entries in the *cam* table of the NAVdb. Additionally, it collects CDP data from all switches and routers supporting this feature; the result is saved in the *swp\_netbox* table for use by the *network topology discover* system.

While its basic operation remains the same, it has been rewritten to take advantage of the OID database; the internal data collection framework has been unified and all devices are treated in the same manner. Thus, data collections are no longer based on type information and a standard set of OIDs are used for all devices. When a new type is added to NAV the cam logging should “just work”, which is a major design goal of NAV v3.

One notable improvement is the addition of the *interface* field in the *swport* table. It is used for matching the *CDP remote interface*, and makes this matching much more reliable. Also, both the *cam* and the *swp\_netbox* tables now use *netboxid* and *ifindex* to uniquely identify a swport port instead of the old *netboxid*, *module*, *port*-triple. This has significantly simplified swport port matching, and especially since the old *module* field of *swport* was a shortened version of what is today the *interface* field, reliability has increased as well.

#### 5.5.4 Network topology discovery

The network topology discovery system automatically discovers the physical topology of the network monitored by NAV based on the data in the *swp\_netbox* table collected by the cam logger. No major updates have been necessary except for adjustment to the new structure of the NAVdb; the basic algorithm remains the same. While the implementation of said algorithm is somewhat complicated as to gracefully handle missing data, the following is a simplified description:

- We start with a *candidate list* for each swport port. These are the switches located behind a switch port and the goal of the algorithm is to pick the one to which it is connected directly. Some of the candidate lists, those of the switches one level up from the edge, will contain only one candidate. We can thus pick this as the switch directly connected and proceed to remove said switches from all other lists. After this removal there will be more candidate lists with only one candidate, and we can apply the same procedure again.
- If we have the complete information about the network we could now simply iterate until all candidate lists were empty; however, to deal with missing information we sometimes have to make an educated guess of which is the directly connected switch. The network topology discover system makes the guess by looking at how far each candidate is from the router and how many switches are connected below them, and then try to pick the one which most closely matches the current switch.

In practice the use of CDP makes this process very reliable for the devices supporting it, and this makes it easier to correctly determine the remaining topology even in the case of missing information.

#### 5.5.5 Vlan discovery

After the physical topology of the network has been mapped by the network topology discover system it still remains to explore the *logical topology*, or the VLANs. Since modern switches support *trunking*, which can transport several independent VLANs over a single physical link, the logical topology can be non-trivial and indeed, in practice it usually is.

The vlan discovery system uses a simple top-down depth-first graph traversal algorithm to discover which VLANs are actually running on the different trunks and in which direction. Direction is here defined relative to the router port, which is the top of the tree, currently owning the lowest gateway IP or the virtual IP in the case of HSRP. In addition, since NAV v3 now fully supports the reuse of VLAN numbers, the vlan discovery system

will also make the connection from VLAN number to actual vlan as defined in the *vlan* table for all non-trunk ports it encounters.

A special case are *closed VLANs* which do not have a gateway IP; the vlan discovery system will still traverse these VLANs without setting any direction and also creating a new VLAN record in the *vlan* table. The NAV administrator can fill in descriptive information afterward if desired.

The implementation of this subsystem is again complicated by factors such as the need for checking at both ends of a trunk if the VLAN is allowed to traverse it, the fact that VLAN numbers on each end of non-trunk links need not match (the number closer to the top of the tree should then be given precedence and the lower VLAN numbers rewritten to match), that both trunks and non-trunks can be blocked (again at either end) by the *spanning tree protocol* and of course that it needs to be highly efficient and scalable in the case of large networks with thousands of switches and tens of thousands of switch ports.

## 5.6 Problems

As already mentioned in the section on `getDeviceData`, a large amount of developer effort went into its current form. In particular, the final design of the OID database, one of the more significant new features of NAV v3, required much discussion on the NAV developer mailing list before all details were worked out to be as good and flexible as possible. In the end we certainly believe it was well worth the effort.

As with any major software development project, a number of smaller technical challenges were encountered during implementation; however none were major and thanks to good communication, especially on the NAV developer mailing list all were resolved within reasonable time.

The biggest challenge has been time, which, despite continuous improvements in the tools used, is common for software development; one reason of course being that more is expected in less time. Despite delays because of the above mentioned longer than expected development time for the OID database and `getDeviceData` it now appears an operational NAV v3 will go into service slightly delayed.

## 5.7 Further Work

NAV v3 is a major rework from previous NAV versions, and much effort has gone into generalizing and creating a framework on which further improvements can be built more easily. The frequent use of plugins is an example of this, and much functionality has been separated out into independent modules which facilitate code reuse. Example modules include

SNMP data collection, sending and receiving of events, accessing NAVdb and storing arbitrary information about a netbox; with the use of the appropriate modules all of these tasks will typically only require one or two extra lines of code in the client program.

In the next version of NAV we can really begin to benefit from this framework in that new features are much easier to create, and we can try out more experimental features. Some ideas which will in all probability be found in the next NAV:

- Cam-logger converted to `getDeviceData` plugin; this will allow much better control of scheduling, e.g. different scheduling for different types.
- Plugins that collect more inventory data from the routers and switches.

Some more long-term ideas which may appear in future NAV versions depending on demand:

- `getDeviceData` plugins for collecting more statistics. Since `getDeviceData` works in parallel the only limitation on the amount of data collected is processing resources and network bandwidth; as both of these increase with time we can also collect more data. Ideas include:
  - Data volume from all switch ports (we do not collect data from EDGE switches today).
  - Real-time statistics (collected every N second) from a set of netboxes on request, for viewing in `vlanPlot` for example.
  - Collecting traps / other statistics from devices like the status of fans, temperature, reboots, and also from external sensors for temperature, air humidity, presence of water etc.

This is just a short list of things we can think of now; the most interesting things are those which we have not yet thought of. Computer networks are, like the rest of the computer industry, always in a state of rapid change and progress, and thanks to the general framework we have created we can be sure NAV will be able to evolve with them!

## 5.7 Concluding Remarks

A major goal of the new data collection system in NAV v3 is to minimize the need for maintenance while better modeling real, modern networks; things should “just work”. We believe this goal has largely been archived based on the progress and improvements described in the preceding sections.

The OID database, a feature long in the coming, and its implementation in `getDeviceData` minimizes the effort necessary to support the introduction of new types of network equipment as much as possible; in many cases almost

no effort is required at all, and most significantly it can be handled without the involvement of the NAV development team.

The enhancements to the *cam logger*, the *network topology discovery* system and the *vlan discovery* system fulfills the second goal of more closely modeling modern networks with the full support of reuse of VLAN numbers.

A large numbers of development hours have gone into the realization of the described improvements, but ultimately we feel the added value to NAV administrators and reduced maintenance for NAV developers will significantly outweigh the cost of development.

## 6. New front end subsystems

### 6.1 Resources

Subproject number	5
Subproject leader	Gro-Anita Vindheim
Developers	Hans Jørgen Hoel
Hours	Budget: 260 Used: 345
Objective achieved	70%

### 6.2 Main objective

- Introduce a web based front end to NAVdb. In other words, replace the cumbersome seed text files of NAV v2. Completed.
- Internal and external status page: Create a more flexible status page. The internal status page with user preferences is 100% completed. The external page is postponed.
- Device Management: Create a system for managing a physical device's life, from arriving to the stock, through the operating phase, and until the device is outdated.

### 6.3 Results

#### 6.3.1 editDB: A web based front end to NAVdb

Previous versions of NAV are based on seeding the database from text files which have been manually updated. tigaNAV introduces a web based interface to the database, where the NAV administrator can add, edit and delete network devices and other "NAV information", like vlan descriptions, organizational units etc. Figure 10 shows the edit database main page.

The NAV administrator must still update the database manually (there is no auto discovery of network devices), but by using the web interface there are several advantages:

- When inserting a new network device, the registration process will check the SNMP community, and give an error message if the wrong community is given. It will also check the type, making sure NAV can collect data from this type.
- The data collection from a new network device will start immediately, the NAV administrator does not have to wait for the scheduled nightly process.
- Better user interface. The NAV v2 seed text files consist of lines with colon separated fields. There is no syntax checker as you edit the files. Typical errors are missing or misplaced colons. These errors are not



detected until the nightly cron process discovers it. And then again the NAV administrator may not notice the log message. With the new NAV v3 web interface the approach is more bullet proof, we give direct feedback on errors.

- Easier to edit the seed data. As for adding new data, the data can be edited using the same forms.

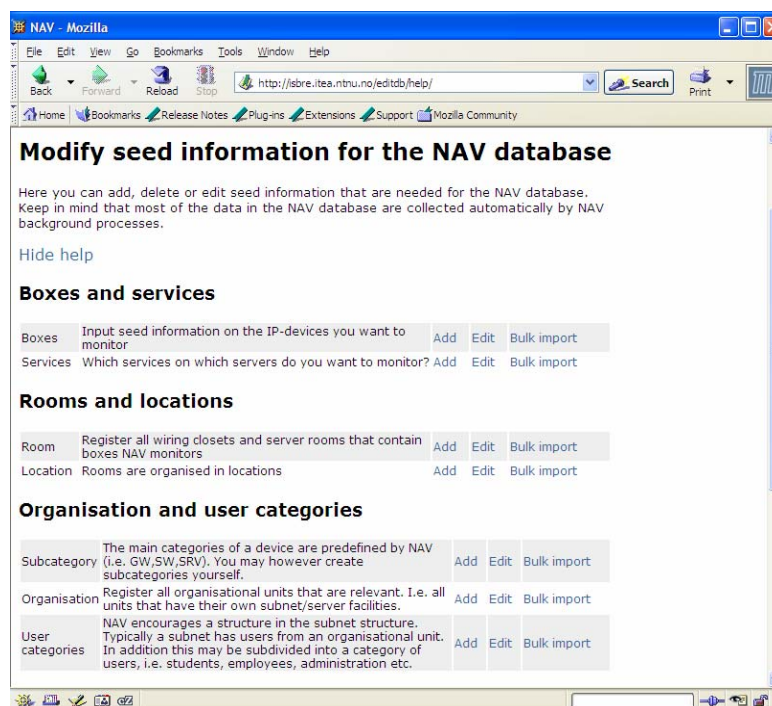
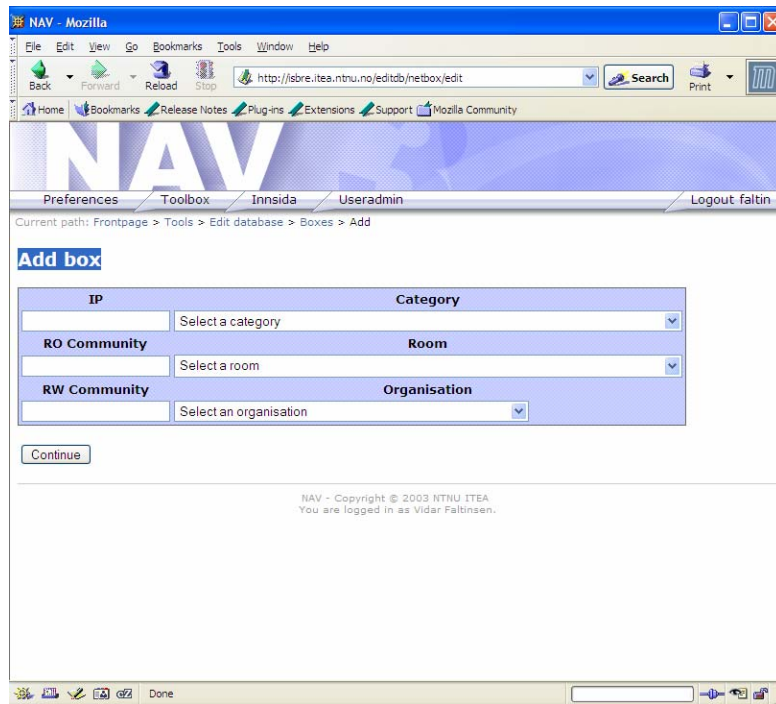


Figure 10: The edit database subsystem

### 6.3.1.1 Add, edit and delete operations

Figure 11 gives an example of a web form for adding a new entry into the database. The shown form is for adding a new netbox (IP device) into NAVdb.

The web interface also makes it easier to edit or delete seed information. Figure 12 shows the web interface for editing and deleting equipment types. The NAV administrator can select one or more types, and then choose the intended action button. The edit button gives a pre-filled form for the NAV administrator to edit, the delete button gives a list of the checked seed data, and asks the NAV administrator to confirm the delete operation.



NAV - Mozilla

File Edit View Go Bookmarks Tools Window Help

Back Forward Reload Stop http://sbrre.itea.ntnu.no/editdb/netbox/edit Search Print

Home Bookmarks Release Notes Plug-ins Extensions Support Mozilla Community

NAV

Preferences Toolbox Innsida Useradmin Logout faltin

Current path: Frontpage > Tools > Edit database > Boxes > Add

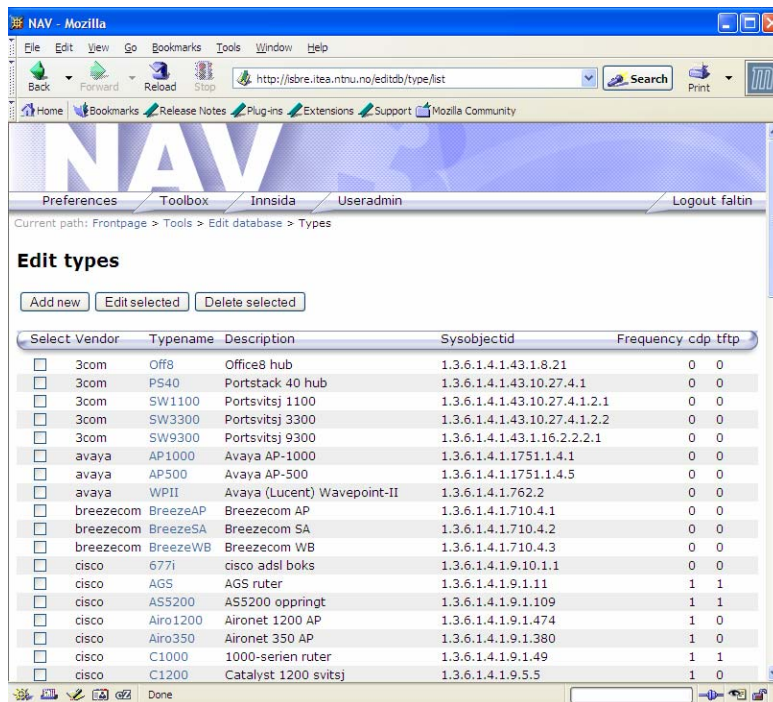
### Add box

IP	Category
<input type="text"/>	Select a category
RO Community	Room
<input type="text"/>	Select a room
RW Community	Organisation
<input type="text"/>	Select an organisation

Continue

NAV - Copyright © 2003 NTNU ITEA.  
You are logged in as Vidar Faltinsen.

Figure 11: Web form for adding a new netbox (IP device)



NAV - Mozilla

File Edit View Go Bookmarks Tools Window Help

Back Forward Reload Stop http://sbrre.itea.ntnu.no/editdb/type/list Search Print

Home Bookmarks Release Notes Plug-ins Extensions Support Mozilla Community

NAV

Preferences Toolbox Innsida Useradmin Logout faltin

Current path: Frontpage > Tools > Edit database > Types

### Edit types

Add new Edit selected Delete selected

Select Vendor	Typename	Description	Sysobjectid	Frequency	cdp	tftp
<input type="checkbox"/>	3com	Off8	Office8 hub	1.3.6.1.4.1.43.1.8.21	0	0
<input type="checkbox"/>	3com	PS40	Portstack 40 hub	1.3.6.1.4.1.43.10.27.4.1	0	0
<input type="checkbox"/>	3com	SW1100	Portsvitsj 1100	1.3.6.1.4.1.43.10.27.4.1.2.1	0	0
<input type="checkbox"/>	3com	SW3300	Portsvitsj 3300	1.3.6.1.4.1.43.10.27.4.1.2.2	0	0
<input type="checkbox"/>	3com	SW9300	Portsvitsj 9300	1.3.6.1.4.1.43.1.16.2.2.1	0	0
<input type="checkbox"/>	avaya	AP1000	Avaya AP-1000	1.3.6.1.4.1.1751.1.4.1	0	0
<input type="checkbox"/>	avaya	AP500	Avaya AP-500	1.3.6.1.4.1.1751.1.4.5	0	0
<input type="checkbox"/>	avaya	WP11	Avaya (Lucent) Wavepoint-II	1.3.6.1.4.1.762.2	0	0
<input type="checkbox"/>	breezecom	BreezeAP	Breezecom AP	1.3.6.1.4.1.710.4.1	0	0
<input type="checkbox"/>	breezecom	BreezeSA	Breezecom SA	1.3.6.1.4.1.710.4.2	0	0
<input type="checkbox"/>	breezecom	BreezeWB	Breezecom WB	1.3.6.1.4.1.710.4.3	0	0
<input type="checkbox"/>	cisco	677i	cisco adsl boks	1.3.6.1.4.1.9.10.1.1	0	0
<input type="checkbox"/>	cisco	AGS	AGS ruter	1.3.6.1.4.1.9.1.11	1	1
<input type="checkbox"/>	cisco	AS5200	AS5200 oppringt	1.3.6.1.4.1.9.1.109	1	1
<input type="checkbox"/>	cisco	Airo1200	Aironet 1200 AP	1.3.6.1.4.1.9.1.474	1	0
<input type="checkbox"/>	cisco	Airo350	Aironet 350 AP	1.3.6.1.4.1.9.1.380	1	0
<input type="checkbox"/>	cisco	C1000	1000-serien ruter	1.3.6.1.4.1.9.1.49	1	1
<input type="checkbox"/>	cisco	C1200	Catalyst 1200 svitsj	1.3.6.1.4.1.9.5.5	1	0

NAV - Copyright © 2003 NTNU ITEA.  
You are logged in as Vidar Faltinsen.

Figure 12: Form for editing or deleting equipment types

### 6.3.1.2 Adding a new IP device (netbox)

As mentioned, most of the seed data insertions are straight-forward. This does *not* include adding new IP devices (netboxes). Figure 13 shows the flow diagram for adding a new IP device. We will comment on several important aspects of the diagram:

- Category

Any device you add to NAV must be placed in an appropriate category. The categories are slightly adjusted in tigaNAV (only minor changes). We now have seven categories, five of these require SNMP support:

Cat	Description	SNMP required
GW	A router. Operates on layer 3 (IP layer).	Yes
GSW	A router and switch in one device. Operates on layer 2 and 3. A given port may be switched or routed. Ex. is Catalyst 6500 in native mode or Catalyst 4500 sup4.	Yes
SW	A switch. Operates on layer 2 (Mac layer). The SW category was originally intended for core switches that had vlans and trunking. If you prefer all switches may be defined as category SW, also those that are in the EDGE-group.	Yes
EDGE	Edge-equipment. Operates on layer 2 (mac layer). Physically a hub/hub stack or a switch/switch stack where we do <b>not</b> use vlans. Distributes traffic to the end-users (in some cases an edge-switch may forward traffic to another edge-switch).	Yes
WLAN	Wireless equipment, i.e. base stations, wireless bridges. Operates on layer 2 (mac layer).	Yes
SRV	Servers. Subcategories are frequently used here.	No
OTHER	Other equipment. Left over, everything else. I.e. vpn concentrators, terminal servers etc. This equipment is monitored by the status monitor, but NAV does not do any SNMP gathering.	No

- The SNMP requirement

When a new IP device is registered and the selected category requires SNMP, NAV will do an SNMP poll. If the poll fails, the NAV administrator will be notified, and asked to check the SNMP community string. The IP device will not be added to the database.

- Type

The type is derived from the IP device's sysObjectID. If the sysObjectID is not registered in the database, the adding process will be aborted, and the NAV administrator is asked to add the new type with the corresponding sysObjectID to the database (also a web interface, of course).

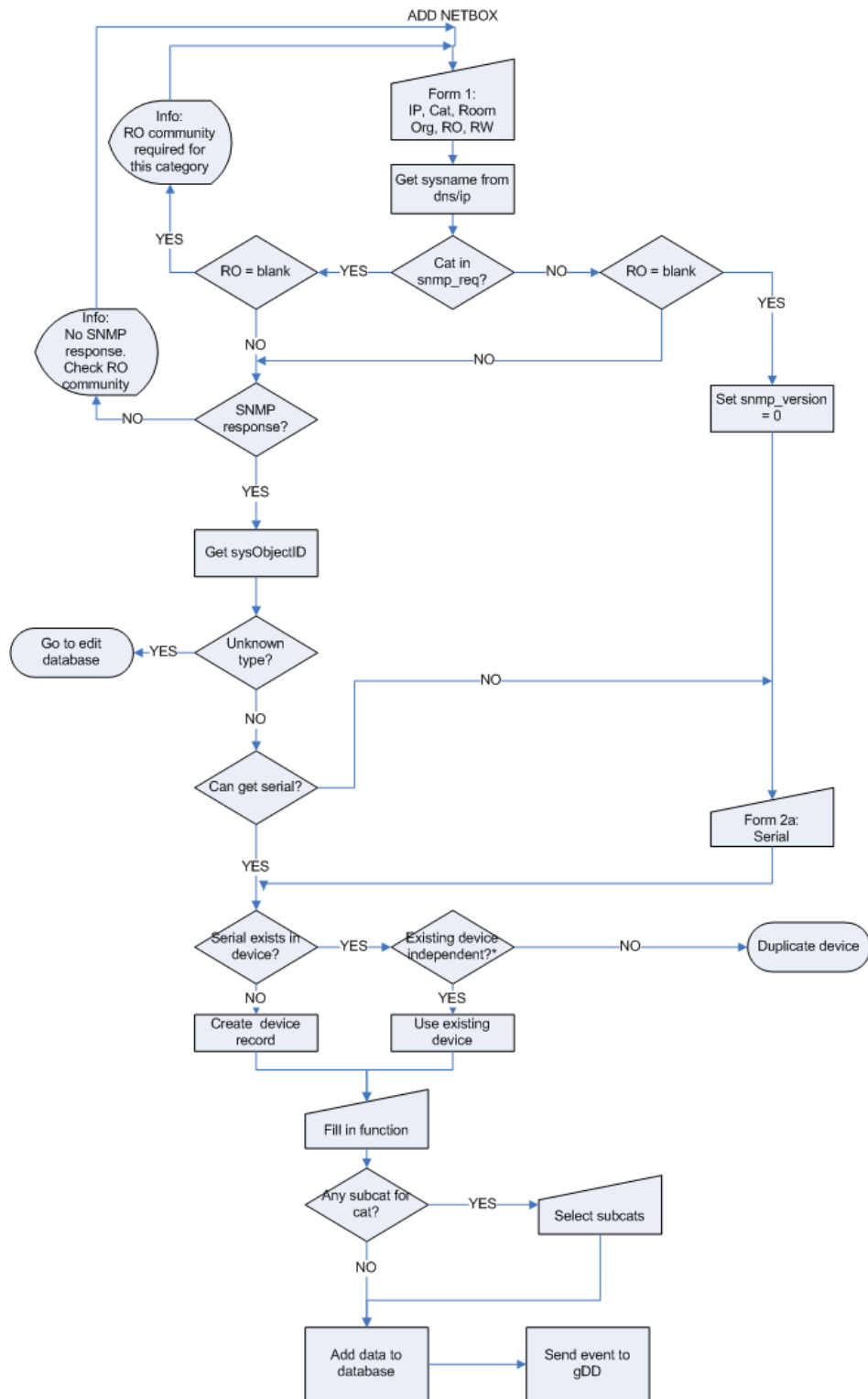


Figure 13: Flow diagram for adding an IP device

- Serial number  
The device's serial number is checked. If it cannot be found through SNMP requests, the NAV administrator will be asked to insert a unique serial number for the IP device. The serial number is then checked against the physical device table in the database. The physical device aspect is described in section 6.3.3.
- Function and subcategories  
The NAV administrator is asked to insert the function and subcategories. These fields are optional.
  - The function is a text string used to describe the operational function of the device in question.
  - The subcategory field can, as the name suggests, be used to classify a device within the category. An example is to classify the servers (category SRV) into the subcategories MAIL and DNS. An IP device be a member of one or several subcategories.

Note that a network device must be up and running before it is added to NAV, due to the SNMP request and type classification. This was not a requirement in NAV v2. We have deliberately changed this to enforce the NAV administrator to enter adequate data when a new device is to be monitored.

### 6.3.1.3 Bulk import

As seen on the edit database index page (figure 10), there is also an option of *bulk import*. This gives the NAV administrator the opportunity to add more than one seed record at a time. The bulk import option can also import files. The bulk import requires a given syntax for the order of the fields, almost similar to the earlier seed data text file format. Bulk import will i.e. be useful when migrating your seed data from NAV v2 to NAV v3.

### 6.3.2 The Status page

The status page (figure 14) gives an overall view of the operational status. The NAV v2 status page showed only netboxes that were down. We have expanded the scope and now give status on:

- netboxes (including shadow reports)
- modules (i.e. modules in a stack of switches)
- services running on servers (i.e. SMTP, http etc)

As new events are implemented, the status page will be expanded to cover these events as well. Next in line are threshold alarms.

Current path: [Frontpage](#) > [Tools](#) > [Status](#)

### Status

#### Boxes down(history)

4 boxes down

Sysname	IP	Down since	Downtime
sit-sby-973-h	129.241.138.32	11:20 25-11-03	7 d, 04 h, 06 m
sit-sby7-938-h	129.241.136.7	12:07 24-11-03	8 d, 03 h, 20 m
kamelon.stud	129.241.56.52	04:06 02-09-03	91 d, 11 h, 21 m
floodhest.stud	129.241.56.24	04:06 02-09-03	91 d, 11 h, 21 m

#### Boxes in shadow(history)

0 boxes in shadow

Sysname	IP	Down since	Downtime
---------	----	------------	----------

#### Modules down(history)

0 modules down, 0 in shadow

Sysname	IP	Module	Down since	Downtime
---------	----	--------	------------	----------

#### Services down(history)

3 services down, 0 in shadow

Sysname	Service	Down since	Downtime
textus4.stud	http	15:21 02-12-03	0 d, 00 h, 06 m
gaupe.stud	http	22:23 17-09-03	75 d, 17 h, 03 m
desperados.itea	imaps	23:20 01-09-03	91 d, 16 h, 07 m

Figure 14: The Status Page

### 6.3.2.1 Status Page Preferences

Since the status page covers so many events, we have implemented a preference option. As indicated on figure 15, a logged in NAV user can set his own preferences for the status page. The user can add, delete and rearrange the sections on his status page. More importantly he can limit his view to his organization (or a set of organizations), to a certain category of devices, or a certain set of services etc.

Add, delete or rearrange sections on your status page.

Name		Filter		
		Organisation	Category	State
<input type="radio"/>	Boxes down Boxes down	All adm alif ark	All GSW WLAN SW	All Down Shadow
<input type="radio"/>	Boxes in shadow Boxes in shadow	All adm alif ark	All GSW WLAN SW	All Down Shadow
<input type="radio"/>	Modules down Modules down	All adm alif ark	All GSW WLAN SW	All Down Shadow
<input type="radio"/>	Services down Services down	All adm alif ark	All dns imaps imap	All Down Shadow

Figure 15: User Preferences for the Status Page

### 6.3.3 Device Management

#### 6.3.3.1 The “device triangle”

So far NAV has only managed devices *in operation*. With tigaNAV we introduce the concept of the “device triangle”. The goal is to have information on the whereabouts of all physical devices at all times. This will include historical information on movements and “milestone events” regarding devices.

We introduce two new tables, named *device* and *module*. Their relation to the *netbox* table is shown in figure 16.

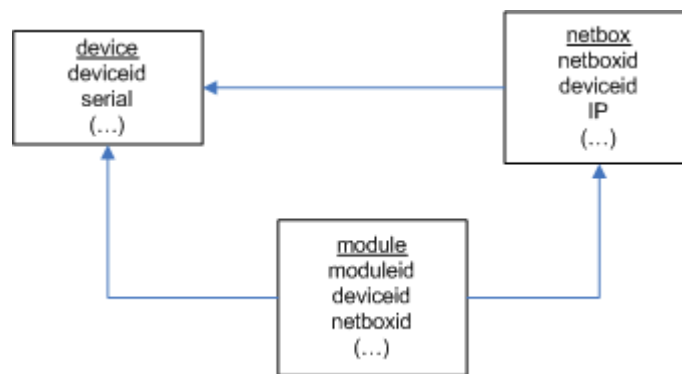


Figure 16: The device triangle: tables *device*, *module* and *netbox*

All physical devices, both operational and on stock, can be registered in the *device* table. The unique identifier is device’s serial number (if the serial number is unavailable another unique labeling scheme is adequate).

A *netbox* (IP device) consists of one or more *modules*. A *module* is physically either a module in a chassis-based switch/router or a switch member of a switch stack. The *modules* are in turn *devices* with unique serial numbers.

In this picture the *netbox* will relate to the physical device (module if you wish) that has the IP-address configured.

#### 6.3.3.2 The Device Lifecycle

Figure 17 illustrate typical processes in a device’s life. The processes can be subdivided into stages; there are 5 stages in a device’s lifecycle:

1. Order: The device is ordered from the supplier.
2. Arrival: The device has arrived on the premises.
3. In operation: The device is put into operation.
4. Production: The device is doing its job.
5. RIP: The job is done.

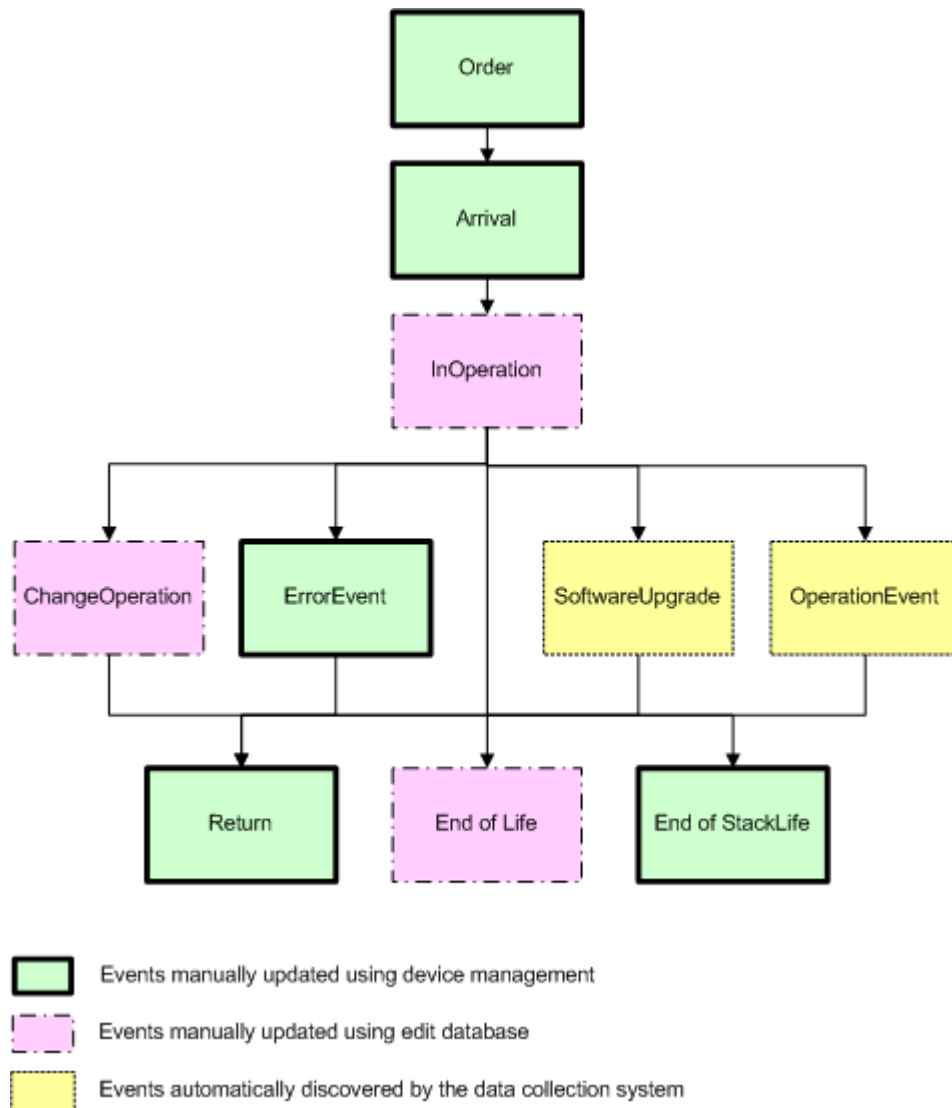


Figure 17: Processes in a device's life

The longest stage is of course production. Here a number of events can occur: software upgrades, temporary (unplanned) outages, special error situations etc. The device may also be reassigned to a different location or to a different IP device (joining another switch stack).

The RIP stage involves return of shipment to the supplier due to fault and warranty. Or more typically, the device has done its job, it can no longer keep pace with traffic development.

The processes trigger events. Events fall into one of two categories:

- *Automatically* detected events: Detected by the data collection system
- *Manually* registered events: Registered using device management or edit database (see chapter 6.3.1)



Figure 18 gives an example showing the life cycle of two switches:<sup>8</sup>

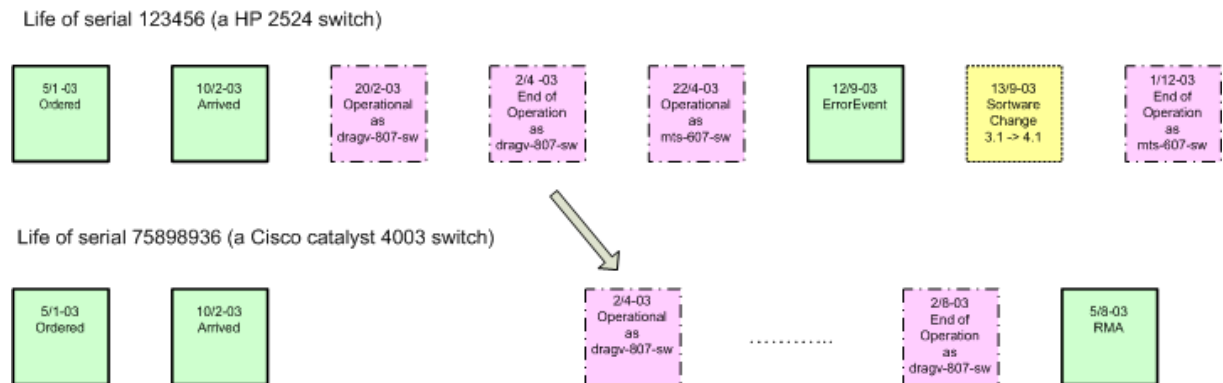


Figure 18: Life cycle of two physical devices

### 6.3.3.3 The Device Management Tool

Device management aims to give the NAV administrator a better way of maintaining all this information. Figure 17 shows which operations the device management tool manages.

Note that it is *not* compulsory to use the order and arrival processes, it is meant as a supplement to those who do not have other logistic systems.

With the device management tool, the NAV administrators can view the history of a physical device, including dates of order and arrival, when it was placed in operation, software upgrades etc.<sup>9</sup>

As shown on figure 17, the NAV administrator can manually register error messages related to incidents with netboxes or physical devices. This information is also displayed in the device history overview.

Finally, device management provides a “manual module delete” option. At times the network engineers will physically remove a module from a chassis/stack. The data collection system will detect this, but has no way of distinguishing a planned relocation from a real outage. The status page will show the event as an outage.

<sup>8</sup> Note that the fill color and border thickness are used in correspondence with figure 17, thereby illustrating which NAV subsystems are involved in the different processes.

<sup>9</sup> The data collection system (chapter 5) will detect changes in software version etc for a given serial number, and trigger an event to Event Engine (chapter 4).

This is where the “manual module delete” option comes in handy. The NAV administrator can manually delete the modules that on purpose are taken out of the chassis/stack in question.

## **6.4 Problems**

The most complicated aspect of Edit Database is the add netbox procedure. Unfortunately we did not design this process thoroughly in the early phases of the project. We had to redesign the flow in October, which of course was unfortunate.

## **6.5 Further work**

We consider the edit database front end to be completed.

The external status page has been postponed. The goal was to create a textual description of the outage, understandable for the common man. Before this external page can be created, the service and network dependencies must be further investigated.

The device management pages need to be restructured, to make the most out of the device management tool.

## 7: RRD Activities

Subproject number	6
Subproject leader	John Magne Bredal
Developers	John Magne Bredal, Erlend Mjaavatten
Hours	Budget: 440 Used:
Objective achieved	75%

### 7.1 Main Objective

- Implement a RRD-database with complete overview of the RRD-files that are created and used in NAV-v3 (completed).
- Enhancement of the script that makes the config-tree for Cricket, for instance by using the RRD- and OID-database that are new in NAV-v3 (completed).
- A better and more flexible way to view graphs created with the data from RRD-files. (completed).
- Sorted statistics for all RRD-data. (not completed).
- A new threshold monitor that is able to adjust threshold values on each and every data source. (completed).
- Large scale Cricket test on a dedicated server. (not completed).
- Workaround for the retrieval of data to the network load map (vlanPlot) so that we no longer see no load. (not completed).

### 7.2 Results

#### 7.2.1 RRD database

The RRD database is currently functional and in use by the implemented subsystems. The layout of the database is shown in figure 19. As we see database contains two tables; `rrd_file` and `rrd_datasource`.

Each RRD file is registered in the `rrd_file` table, along with the path to the file, the step in seconds for how often it is updated, the netbox it is associated with and the subsystem that is currently updating the file. The key and value fields are optional in use and may for instance be used to point to a field in another table that tells in more detail what type of data is stored here.

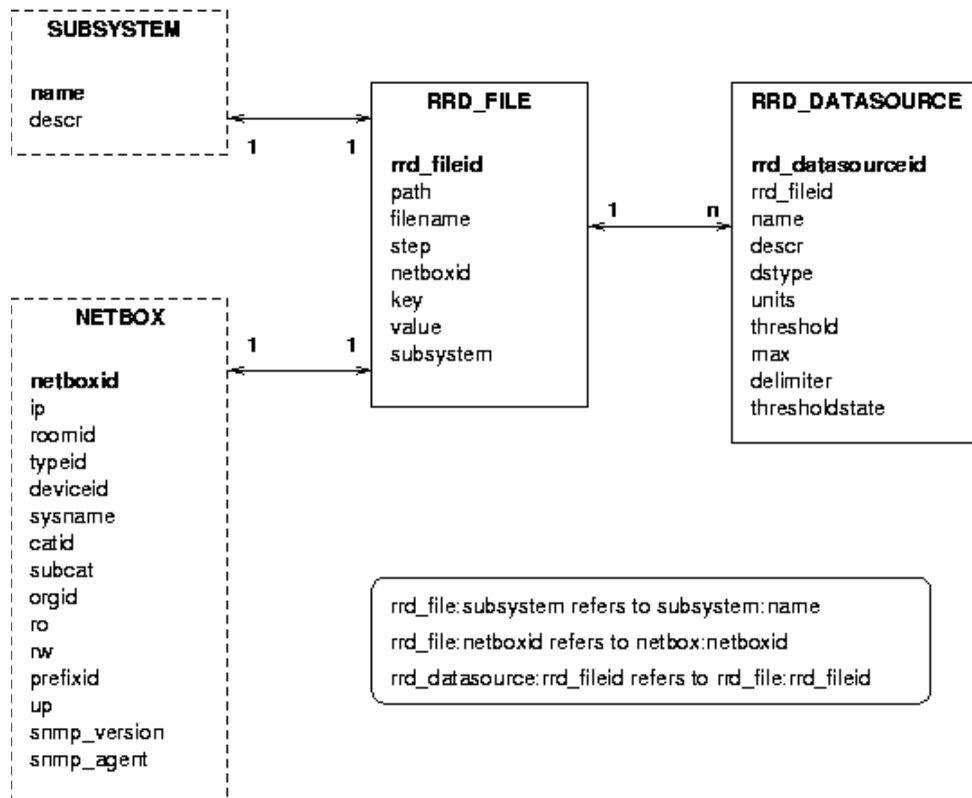


Figure 19: The RRD database

In `rrd_datasource` we store every data source that we collect data for in a `rrd_file`. The name and description field tells what is stored here. `dstype` tells us what type this datasource is, as we use RRDTool to create the RRD-files. `units` is supposed to say what units are to be used on the y-axis of the graph (seconds, bytes and so on). The last four fields (`threshold`, `max`, `delimiter` and `thresholdstate`) is used by the threshold monitor.

### 7.2.2 makecricketconfig

`makecricketconfig` is the script that makes the config-tree for Cricket. It has been active since the start of NAV-development and has gone through a series of enhancements. In NAV-v3 the following enhancements have been made:

- Total rewrite of script to get rid of a messy structure which has been the result of a number of additions to the script.
- The use of `.nav` config-files in the config-directory structure. These files help `makecricketconfig` take the right decisions when building the config-tree.
- Use of RRD-database for storing of RRD-information.
- Use of OID-database to make correct configuration for the different network equipment types.
- No more need for a dedicated tree of default config-files. We now edit the default config-files directly.

### **7.2.2.1 The .nav-config files**

The .nav-config files help makecricketconfig make the right decisions when building the config-tree. The files are hidden (they start with a dot). This is intentional because otherwise the compile-script that Cricket uses will think that the file is a part of its configuration during compile time.

The first .nav file is in the top of the config-tree. This file tells makecricketconfig what directories to make config for. In each of these directories there must be another .nav-file. This file in turn tells makecricketconfig what kind of config is to be stored here, for which type of network equipment. The file also has options for the naming of the files and how to make a description of each view. This is in great aid especially when making configuration for interfaces.

### **7.2.2.2 Use of the RRD database**

The RRD-database is described earlier. The use of this database will make finding and access to RRD-files easier than before. makecricketconfig fills this database with the information it has after building the config-tree. It will not delete tuples in the database, as these files may still exist even though they are not present in the config. The reason for this is that a device may temporarily not respond when making the config-tree, and we do not want to lose information stored in the RRD-file just because of that. Another script will take care of the deletion of unused RRD-files in the database.

### **7.2.2.3 OID database / direct editing of Default-files**

The OID database is used in the making of the config-tree. Based on the information there, we edit the Cricket Default-files and make target types for the different types of network equipment. This will minimize errors from Cricket based on not being able to find the correct OIDs on the network equipment during collection of data. Minimizing errors also minimize the time used for collection of data. This is one of the main improvements of the script.

## **7.2.3 A better and more flexible way to view graphs**

Cricket presents us with a directory structure when viewing graphs. This is not the best way to easily select the graphs you want to see, and may lead to an unnecessary series of clicking before the wanted results are presented. This has been improved in NAV-v3. The way this has been done is by linking the graphs from other subsystems of NAV-v3. In that way you can see the graphs you want to see from i.e. the Device Browser. In this way we don't need a special browser for this purpose, we just use other subsystems. And if users want to use the old Cricket interface, then this is of course also possible.

A special module has been made to simplify access and graphing of information from RRD-files. This module is written in Python and offers an interface both to other scripts and to the web-pages (as they are written in Python too). The module offers the following methods:

- access to numerical values from RRD-files.
- simple mathematical operations on the data
- generation of graphs
- dynamical addition and removal of datasources to the graphs
- generation of url's to graphs generated by the module

The RRD-browser, documented in section 9.3.5, makes use of this Python module.

#### 7.2.4 Sorted Statistics for all RRD data

This subproject has not been completed. There are however work in progress.

#### 7.2.5 Threshold Monitor

A new and improved threshold monitor has been made. The work with this monitor was simplified enormously by the python module mentioned in section 7.2.3 above together with the RRD database.

The threshold monitor uses four fields in the RRD-database for storing of different information about thresholds. All the fields are in the `rrd_datasource` table.

- `threshold`: the threshold value to be checked on this data source. This may be stored as an integer or a percentage number.
- `max`: the max value that this data source may be. This is only important if the threshold value is stored as a percentage, in which we use this field to calculate the percentage in use (see example below).
- `delimiter`: tells us if the value must be higher or lower than the threshold to trigger an alert. The delimiter is either `<` (less than) or `>` (greater than).
- `threshold state`: if this is set to "active" we have sent an alert on this data source, if it is inactive we have not. In an ideal world every threshold state is set to "inactive".

### Example of use of the fields:

For each datasource where threshold is set:

```
if threshold is percentage:
    value = value_from_file * 100 / max
else:
    value = value_from_file

if value delimiter (< or >) threshold and thresholdstate is
inactive:
    set thresholdstate to active
    send activealert
if not value delimiter (< or >) threshold and thresholdstate
is active:
    set thresholdstate to inactive
    send inactivealert
```

The threshold for sending an inactive alert is adjusted by a certain percentage of the active alert threshold. The reason for this is that we do not want to send a series of active/inactive alerts when the value is going up and down in the threshold-area.

### 7.2.6 Large Scale Cricket Test

On networks the size of NTNU's we have a lot of network equipment. As statistical data of this equipment may help the troubleshooting of various network anomalies, it is desirable to have as much statistics as possible. As the number of units to gather data from increases, this of course takes more and more time. Cricket gathers by default data every fifth minute. This means that we must be able to gather all the data we want within five minutes, or some data may be lost or become inaccurate. At the moment we are not able to collect data from all the equipment we want to because of this.

There is a simple solution to this, and it is called parallelization. The only problem is that by doing multiple collections at the same time, we create a higher demand for CPU power. This demand is so high that prior tests on NAV-installations show that we need a dedicated computer for this task. More CPU power means more parallel collections and more data in those five minutes.

As this has not been tested we need to do a full scale test on NTNU's network to see how much data we are able to collect in those five minutes. As we regard our network to be fairly large, this will be a good indication of how much power that is needed in any installation of NAV.

This task has not been done, and is not planned in nearby future. As this is not a critical task in NAV-v3, it is postponed until time and equipment is available.

### 7.2.7 Reliable Collection of Data to the network load map

This task is postponed as the network load map (vlanPlot) is not yet ported to NAV v3.

## 7.3 Problems

makecricketconfig is dependant on a well-working database to make a correct config-tree. There are situations, which normally should not occur, where the script will mess up the Cricket Default files. Future versions of the script should be able to avoid these situations.

## 7.4 Further Work

- There is currently no automatic making of views for the target types that are made by makecricketconfig. This is a major drawback and must be fixed before a beta-launch of the system.
- At the moment, the threshold monitor does not send alarms to the event engine. This is of course an extreme drawback and must be fixed before a launch of the system.
- The script that deletes unused RRD-files is not functional. There are however only minor details that lack to make a working copy of this script.



## 8. Enhanced Message of the Day

### 8.1 Resources

Subproject number	7
Subproject leader	Gro-Anita Vindheim
Developers	Bjørn Ove Grøtan
Hours	Budget: 120 Used: 80
Objective achieved	75%

### 8.2 Main objective

The main objective of this subproject is to provide a message system the NAV operators can use to inform about planned outage, and inform during problem solving of unplanned outage. The NAV users will also be able to subscribe to these messages, by mail or SMS.

An RSS feed will also be provided by the message system, to enable other user interfaces (such as Innsida, NTNU specific) to import the information.

During the project, the message system evolved to include a “put on maintenance” option, setting a room (wiring closet), network device or service (ssh, web) on maintenance for a given timeframe. If the surveillance system reveals outage during the device’s maintenance timeframe, the NAV users will not be notified.

### 8.3 Results

As described in the previous section, a room (wiring closet), a netbox (IP device) or a service can be set on maintenance. For simplicity, this chapter will use the term *unit* when referring to one of these.

#### 8.3.1 Message of the day

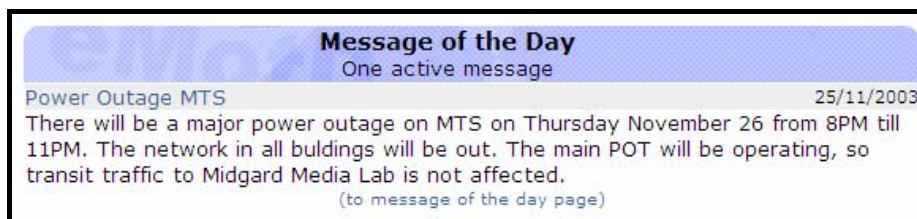
The main page for inserting messages supports two languages, Norwegian and English. It is an ordinary web form, where the NAV operator fills in the appropriate data, such as title, short description, technical description, affected users and timeframe. There is a checkbox to indicate whether one or more units are to be set on maintenance in connection with the message inserted.

A message can be of one of three types:

- **Informational.** Message intended to tell the users about a future change, possibly with a due date and a description about how the changes affect the users.

- **Error.** Message intended to tell the users about error situations like unplanned outage, and what is being done to fix it.
- **Internal.** Message intended for internal use only. It will not be shown on the NAV front page unless the NAV user is logged in.

All active messages are shown on the web during their given timeframe. An example is shown in figure 20.



*Figure 20: Example of eMotD message as shown on the NAV main page*

The NAV operator can supply more information to an existing message by writing a follow-up message. The original message will then be outdated, and the message system will show the new message.

The messages can be imported into other medias, using the defined RSS feed.

### 8.3.2 Set on maintenance

By attaching the maintenance function to the message system, we force the NAV operator to give a textual explanation of why a unit is set on maintenance, and for how long. This prevents units being “on maintenance” for weeks (or months).

The maintenance timeframe can be updated from the web interface. The NAV administrator may for instance realize that the timeframe was set too short (or too long). Sometimes a planned outage has to be postponed for some reason.

The maintenance tool page also provides a view displaying all units with a scheduled outage for the next 24 hours.

### 8.3.3 Database design

The eMOTD database design consists of three tables, as shown on figure 21.

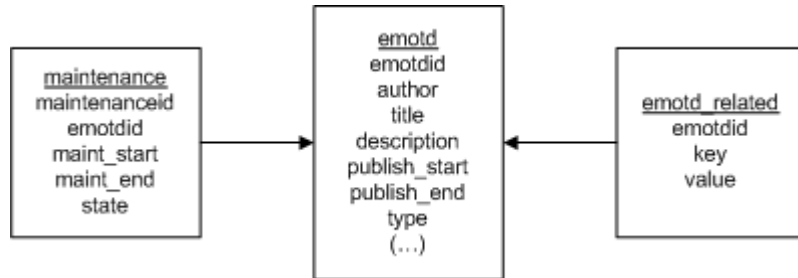


Figure 21: The eMOTD database design

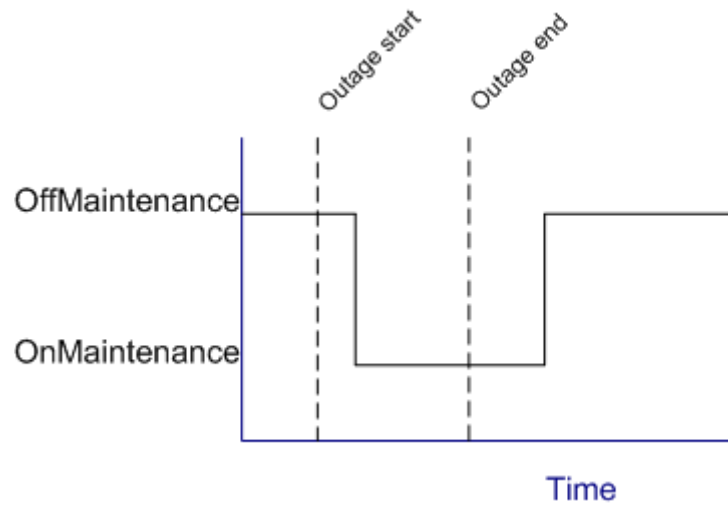
The *emotd* table is the primary table, containing the message title, body, published timeframe etc. The *maintenance* record is linked up to a given *emotdid*, defining the maintenance timeframe and current state (active, overridden, etc).

The *emotd\_related* table keeps track of the units put on maintenance related to a given *emotdid*. The *key* field defines the table name (i.e. *netbox*), and the *value* field defines the primary key value in the table given by *key*. Note that several *emotd\_related*-records can be linked up to one *emotdid*. When updating the maintenance timeframe for an *emotd* record, it will be updated for all attached *emotd\_related* records, because the timeframe is set in the maintenance record only.

### 8.3.4 Background processes

As part of the maintenance system, there is a background process that updates the state field in the maintenance table (see figure 20). The background process also posts events to Event Engine. Event Engine will, based on events from the maintenance background process, decide whether the NAV users are to be notified of a given outage. Read more about the Event Engine in section 4.3.1.

If a unit's outage exceeds the maintenance timeframe, the NAV user can experience receiving only one of the outage alarms. Figure 22 shows an example where the planned outage starts before the predefined maintenance timeframe. The NAV users will receive a boxDown alarm, but they will not be notified when the device is operational again. We are aware of the problem, but have no ideal solution at the moment.



*Figure 22: Example of inconsistent outage and maintenance timeframe.*

## 8.4 Further work

The message system has all basic functionality. Further work should focus on the user interface, making it more user friendly.

## 9. Device Browser

### 9.1 Resources

Subproject number	8
Subproject leader	Morten Vold
Developers	Magnus Nordseth, Stian Søliland
Hours	Budget: 400 Used: 430
Objective achieved	75%

### 9.2 Main objective

The goal of the device browser subproject was to get a unified starting point for examining a device, its relations to other devices and a short view of other information available on the device.

The idea is that the NAV user could start by visiting the specific device he had in mind, clicking further up or down on what might be interesting. The page should provide basic information on the device and links to more details, both within the device browser itself and other parts of NAV.

### 9.3 Results

#### 9.3.1 General view

A device browser page for a netbox varies according to what information is available, type of device, etc. The look-and-feel should be rather consistent between different views, though, although it might not always be clear for the user why some information is not available and displayed. We have however currently decided that this is a better solution than having a lot of fields with "No information available".

An example is given in figure 23. As shown the device browser displays basic information for a given netbox, like current status (up/down), availability (ping times), IP address, category, localization and type. All grouping parameters, like category and room, provide links to a report of netboxes in the given set. Availability percentages links to the RRD browser.

Recent alerts from alerthist are listed, however there is currently no link to 'All alerts'.

If there are known RRD statistics for this netbox registered in the database, (usually from cricket), they are listed with links to the RRD browser.

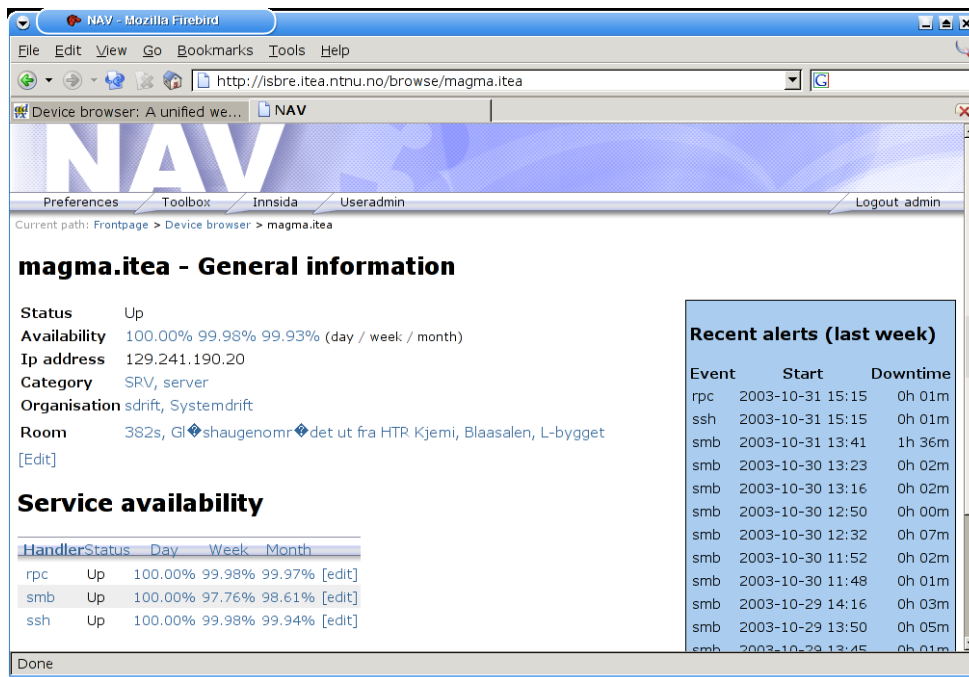


Figure 23: The Device Browser

### 9.3.2 Services

For a server, services monitored on the server are listed. For each service, availability statistics are provided, these are links to the RRD browser as well. In addition, selecting a service handler displays statistics for all monitoring on the given service. Such tables are sort able, in this example services are sorted according to availability.

The screenshot shows the NAV Services view in the device browser. The address bar displays <http://isbre.itea.ntnu.no/browse/magma.itea>. The page title is "All servers running smb". The current path is "Frontpage > Device browser > Services > smb".

**All servers running smb**

Server	Org	Status	Day	Week	Month	
kabul.stud	Systemdrift	Up	100.00%	99.99%	100.00%	[edit]
sarajevo.stud	Systemdrift	Up	100.00%	99.99%	99.99%	[edit]
roma.stud	Systemdrift	Up	100.00%	99.99%	99.99%	[edit]
washington.stud	Systemdrift	Up	100.00%	99.99%	99.99%	[edit]
paris.stud	Systemdrift	Up	100.00%	99.96%	99.96%	[edit]
bagdad.stud	Systemdrift	Up	100.00%	99.95%	99.95%	[edit]
tokyo.stud	Systemdrift	Up	100.00%	99.96%	99.95%	[edit]
lima.stud	Systemdrift	Up	100.00%	99.93%	99.94%	[edit]
magma.itea	Systemdrift	Up	100.00%	97.76%	98.61%	[edit]

NAV - Copyright © 2003 NTNU ITEA  
 You are logged in as NAV Administrator.

Figure 24: Services view in device browser

There is a 'grand total' view of services giving a quick view to the current status as a matrix of netboxes and services (see figure 25). This matrix could be extended to include monitoring of services that should *not* be present. (Currently a subproject at ITEA systemdrift).

Netbox	Ssh	dns	ftp	http	imap	imaps	pop3	rpc	smb	smtp	ssh
alfa.itea	Up										
bagdad.stud								Up			
bb.itea		Up									Up
cassarossa.samfundet.no		Up							Up		Up
cirkus.samfundet.no	Up	Up									Up
desperados.itea			Up	Down	Up				Up		Up
due.stud			Down	Down	Down				Down		
edder.stud											Up
elefant.stud							Down				Down
fagweb.ntnu.no		Up									
flaske.stud			Up	Up	Up				Down		Up
gaupe.stud		Down									Down
illac.stud											Up
jeeves.stud											Up
kabul.stud								Up			
kamel.stud											Up
kiwi.stud											Up
kryptonitt.felles	Up										
leopard.stud		Up									Up
lima.stud								Up			Up
ludvig	Up										Up
magma.itea							Up	Up			Up
ozelot.stud											Up
panda.stud											Up
paris.stud								Up			Up
post.stud			Up	Up	Up				Down		Up

Figure 25: The services matrix

Links to editDB are provided for both the netbox and the services, although there is currently no easy way to add a service from this interface.

### 9.3.3 Modules and Ports

For network equipment, especially switches, modules and their ports are listed. The device browser makes a simple guess on the layout of the switch to get a nice listing of the ports, see figure 26.

For each port it is possible to see its current status by the graphical legend (although the color scheme is not perfect). By moving the mouse over a port, textual information is shown, like "10 Mbit, half duplex, trunk". Clicking on a port goes to a separate port page showing port information (like vlans and uplinks) and statistics that might be available for the port.



Figure 26: The Device Browser Switch View

### 9.3.4 Proper URLs

We have put an effort into creating URLs that are easy to read, remember and give away. For example:

- a view of the machine magma.itea.ntnu.no would be shown at:

<http://beta.nav.ntnu.no/browse/magma.itea.ntnu.no/>

- information on a given port behind a switch:

</browse/blasal-sw2.ntnu.no/module1/port2/>

- a listing of all netboxes running the service smb

</browse/service/smb/> .

### 9.3.5 RRD browser

At a point, the RRD browser was integrated into the device browser instead of living its own life. The reason for this was that RRD statistics are tightly bound to specific devices, and this gave easier access to providing one-liner-statistics in the device browser. The RRD browser uses the underlying modules documented in 7.2.4. Functionality in the RRD browser includes:

- A mechanism to select a set of RRD datasources and display them all on the same web page.



- Mechanisms to zoom in on a graph, i.e. alter the default scaling that was chosen for you.
- Mechanisms to join two or more graphs in the same graph view. The reverse operation is also supported; you may split combined data in separate graph views. You may also remove RRD sources from your view.

An example is given in figure 27.

We consider the RRD browser immature. tigaNAV has implemented a proof of concept that we would like to elaborate on next year. In our opinion, the RRD browser will be a very powerful tool, where the NAV user can combine the statistics he is interested in one view. In many cases this will give a whole new dimension to interpreting the incident at hand.

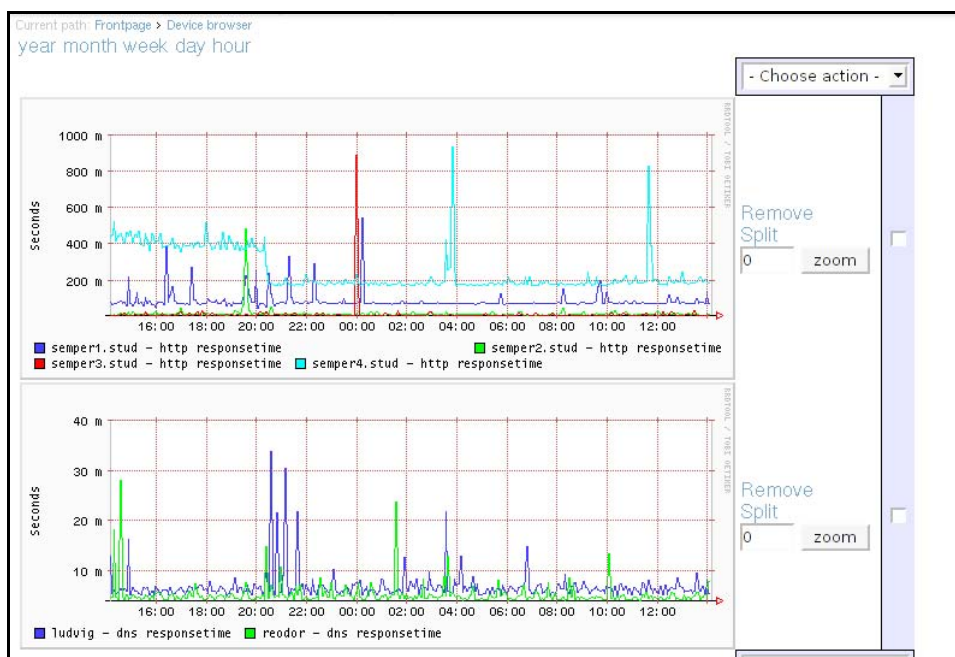


Figure 27: The RRD browser

## 9.4 Problems

During development, what made the most trouble was unreliable data in the database. Displaying modules and ports requires correct representations in the database, but debugging these parts was difficult when `getDeviceData` was in constant development and changing the content of the database all the time.

Without proper tests to run for `getDeviceData`, motivation decreased at severe levels at times. At some phases of the tigaNAV project, even the very late phases, the database design seemed to change all the time, causing all web pages to break repeatedly.

Also, inconsistencies in the database used for development continued to live for a long time. It was decided at an early stage to use full sysnames (ie. `magma.itea.ntnu.no` instead of `magma.itea`), but this was never fixed in

the database. Likewise, at one point the tigaNAV project decided to go for UTF8 in the database to be consistent with character sets, but this was never fixed as well, giving the rather weird <?>-characters seen on the screenshots of this chapter.

The database had redundant tables and fields that were only relevant for NAVv2, and since they were not removed, they confused us several times when we tried to decide what information to retrieve from the database.

If any conclusions could be drawn from this, it could be to use separate databases while developing, as a changing database scheme and content confuses and causes trouble for other developers. Although this would cause an overhead in integration cycles, it would be the proper thing to do when lacking full unit tests for all parts of NAV.

The real way to go would be to have full unit tests, written by using test driven development (writing tests before code), having free ownership of code (everyone can fix)<sup>10</sup> and proper change management for last-minute-changes of database schemas or APIs.

## 9.4 Further work

- IP-address should link to a VLAN report
- Servers should display which switch port they are on, giving a link to the "mother" switch
- More integration with editDB
- Switch port legends are still not perfect, colors are too alike
- Links to reports should use simpler, easier to understand reports
- More parts of NAV should link to the device browser
- Much of the code should be refactored. This goes specially for: - dispatcher.py - urlbuilder.py
- Service dependencies are not addressed in tigaNAV.
- Integration with eMOTD

## 9.5 Concluding remarks

The device browser has fulfilled a need for getting a “physical” view of a single device in NAV. The browser is still not tested much with actual users (much due to the lack of proper data to show), so it is not easy to define at this point if the device browser will do the job.

---

<sup>10</sup> Note from the project leader: The current policy is that everyone can fix, although this is not widely deployed.

However, the device browser seems like a very good starting point, taking focus away from database specific listings and focusing on what information is available.

## 10. Round Trip and Packet loss

### 10.1 Resources

Subproject number	9
Subproject leader	John Magne Bredal
Developers	Magnus Nordseth
Hours	Budget: 40 Used: 20
Objective achieved	50%

### 10.2 Main objective

This small subproject has had two objectives:

- Make the round trip and packet loss data available through the RRD database (completed).
- Look into Cisco's SAA solution and see if it can be integrated in NAV in some way (postponed / deprecated)

### 10.3 Results

Response time and packet loss data were made available with the pping and servicemon development project in NAVMore. The principle has been to input data directly into RRD during the monitoring operation of the two daemons.

The add-on made in tigaNAV has merely been to update the new RRD database with metadata (see chapter 7.2.1 for more). In addition the statistics have been made available from the Device Browser (chapter 9) and through the RRD browser (ch 9.3.5).

We give two examples in figure 28 and 29.

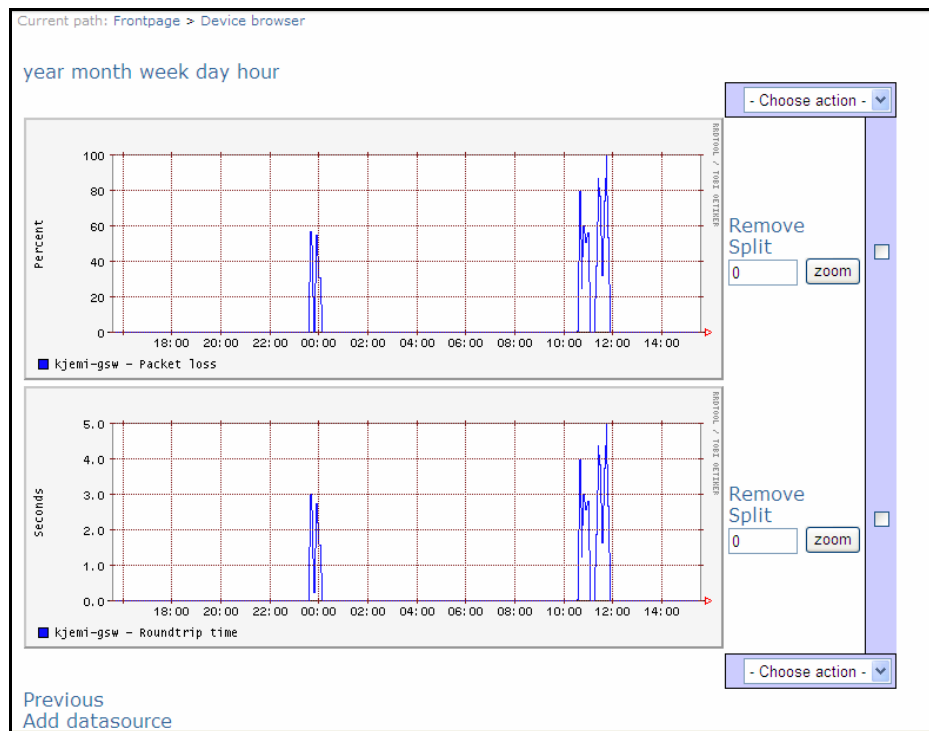


Figure 28: Packet loss and roundtrip time

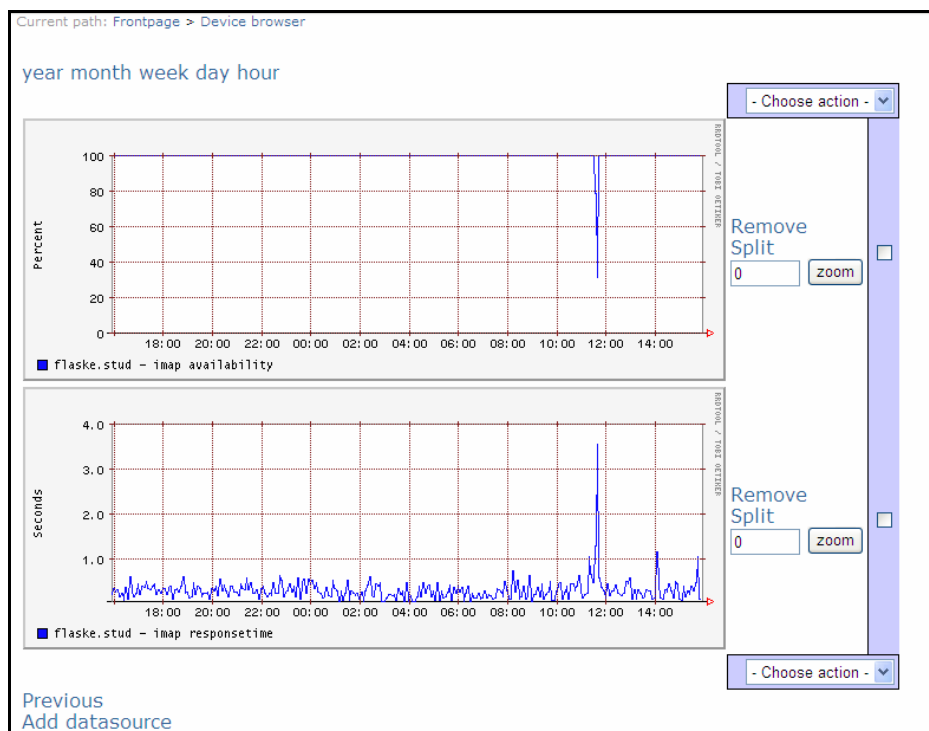


Figure 29: service (imap) availability and response time

# 11. New Network Utilities

## 11.1 Resources

Subproject number	10
Subproject leader	Gro-Anita Vindheim
Developers	Sigurd Gartmann (+ Kristian Eide <sup>11</sup> )
Hours	Budget: 80 Used: 40 (+90)
Objective achieved	30%

## 11.2 Main objective

The main objective of this subproject is to create web interfaces for the following:

- List all machines behind a given switch port at a given time
- Display recently used switch ports
- Give an on-the-fly status for the switch ports of a given switch

We also include the network explorer in this chapter, even though it formally is outside the scope of tigaNAV.

## 11.3 Results

### 11.3.1 Machines behind a switch port

The machine tracker is one of the key features of NAV v2. The ability to locate a machine in the network at a given time (with a history of 30 days) is very important. tigaNAV has ported the NAV v2 solution without altering the basic functionality.

In addition one significant enhancement has been made: Now the machine tracker can display all machines (mac addresses) that have been detected behind a given switch port at a given time.

### 11.3.2 Recently used switch ports and on-the-fly status

The “recently used ports” and the “on-the-fly-status” will both use the switch port view designed by the device browser. Because of this dependency, the features have been postponed until the device browser is ready. We have, however, implemented these two features in a NAV v2 toolkit, though with a different look-and-feel.

---

<sup>11</sup> Development of Network Explorer.

Technically speaking the “recently used ports” is merely an SQL-query in the cam table. “On-the-fly status” is a bit trickier. In the NAV v2 toolkit we do a live SNMP poll on the switch in question and collect the relevant information. For NAV v3 we will implement an “update” button in the device browser’s switch view. An “update” will trigger an event for Event Engine, who in turn will instruct `getDeviceData` to collect the data from the switch in question. The advantage of this approach is that we update NAVdb at the same time. The challenge is how to inform the device browser that the job is done (the response time may vary since we are doing live SNMP polls). One solution is to include a timestamp in the database giving information on the “freshness” of the switch port data.

### 11.3.3 Network Explorer

The network explorer was implemented after NAVMore, prior to tigaNAV. We have a running implementation at NTNU that works with the NAV v2 database. The solution is not yet ported to NAV v3, but this is not expected to require much work.

Network Explorer will complement the network load map<sup>12</sup>. A limitation of the network load map is that it does not give an overall view of the topology of an entire vlan. The NAV user has to click his way from switch to switch and memorize where he has been in order to get the complete picture.

Network explorer, on the other hand, does not display traffic. So it certainly does not replace the network load map. But when it comes to topology we consider the network explorer superior. The tool also has some very useful search options as well.

Figure 30 gives an impression of the tool. The top level shows all the routers in the network as a list (we do not show the interconnection between routers as the network load map does). Each router can be expanded by pressing the plus-button (+). An expanded router shows the second level, i.e. all router ports with corresponding IP prefixes, their description and corresponding vlan number. A given subnet / vlan can be expanded further to show the layer two structure with all interconnected switches. For each switch we display the port number and port name and the connected netboxes, if any.

In addition we have implemented a search tool. The NAV user may search for a netbox (sysname / IP address). If found, Network Explorer will automatically expand the relevant part of the network structure and mark the device you searched for. In fact, figure 30 shows a search for the server named ludvig.

---

<sup>12</sup> The network load map (vlanPlot) is the graphical front end of NAV v2 displaying network traffic and topology in a graphical drill-down view. The network load map is further documented in the NAVMe report (the 2001 NAV development project).

Similarly one can search for a specific switch port name (or part of the name), or all equipment in a room (wiring closet) or all switch ports connected to a vlan. More searches can easily be implemented.

Finally the idea is to have the network explorer link to the device browser at the netbox level. These two tools alone will prove a significant improvement to NAV.

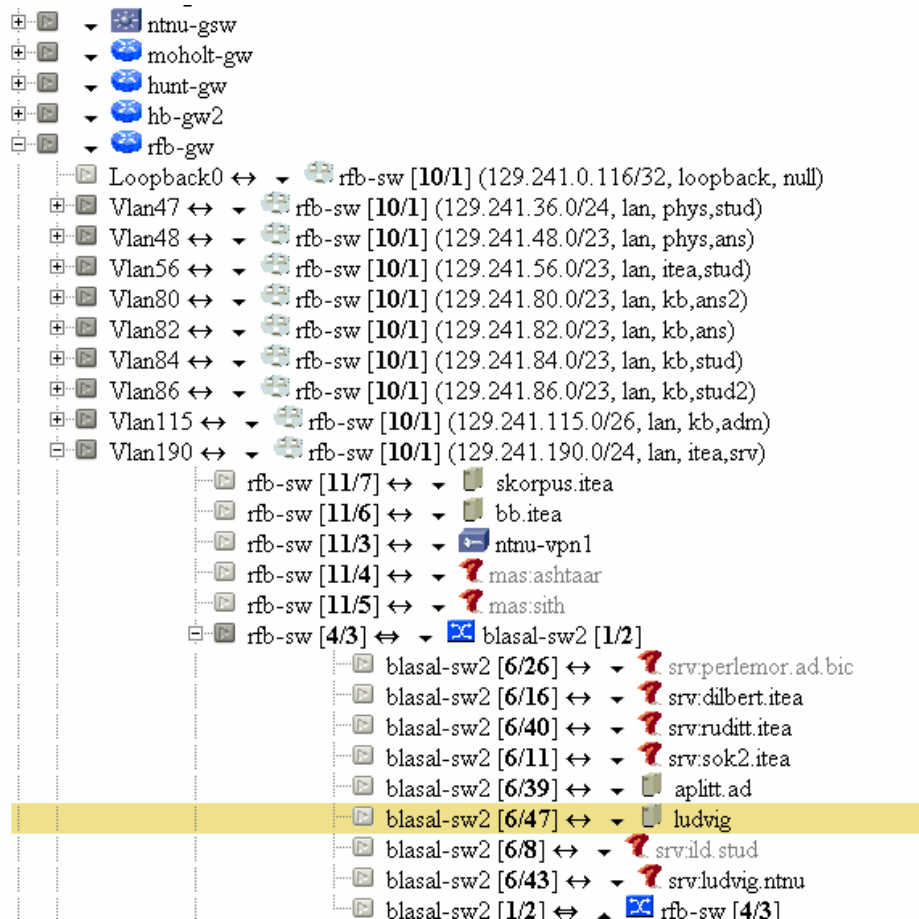


Figure 30: Network Explorer



## 12. Version Control and Software Build

### 12.1 Resources

Subproject number	11
Subproject leader	Morten Vold
Developers	Morten Vold
Hours	Budget: 80 Used: 120
Objective achieved	80%

### 12.2 Main objective

We want to restructure the CVS repository layout from one that mirrors an installed NAV system into one that resembles a typical source code tree. In such a tree, source code is grouped into subsystems, and is more readily available to a software build system.

Also, we wanted to restructure the installation layout of NAV into one that more resembles a typical UNIX directory hierarchy.

A better software configuration/build system for NAV as a whole was needed, to escape from the monolithic one-script know-it-all approach to installing NAV.

### 12.3 Results

#### 12.3.1 CVS vs. Subversion

NAV used CVS as its version control system. CVS is an old hack upon RCS, and although mature and proven, contains many design flaws which prevent it from growing beyond its current limitations. After considering alternatives, it was felt that Subversion would be a worthy replacement. Subversion is aimed specifically at replacing CVS, and has been written from scratch to do so.

As an "improved" RCS, CVS still revolves around versioning single files. One result of this is the lack of atomic commits, something Subversion features from the ground up. Subversion versions entire change sets as one revision.

Another important feature of Subversion is the ability to rename and move files and directories while keeping their history. In CVS, renaming or moving a file means creating a new file and deleting the old file. The new file will have no history of where it came from in the repository. Restructuring NAV's CVS repository would be painful with this restriction, whereas with Subversion it would be trivial.

After some research, Subversion was compiled, installed and tested on bigbud, the server running the NAV CVS repository. After successful testing, the CVS repository was converted into a Subversion repository. The new Subversion repository was networked using Apache+SSL and was operative from July 1st.

The Subversion command line client is so similar to CVS in daily use, that no special training of NAV developers was needed. The new repository was in full use by everyone on day one. There have not been any Subversion-related problems since the conversion, and feedback has been nothing but positive.

### 12.3.2 Restructuring source code repository layout

NAV v3 has introduced many subsystems, both new and rewritten. Most of the subsystems consist of several source code modules, configuration files and documentation. Each new subsystem has been grouped below the subsystem/ directory of the Subversion repository. This structure allows for a developer to find all files related to a particular subsystem in one place, rather than all over the repository.

The Java subsystems inherited from NAV v2 were already grouped under the src/ directory. So far, these subsystems have remained in this directory, and new Java subsystems, such as getDeviceData and event Engine, have also been placed here.

Many outdated and unused files have been removed from the repository, and most existing subsystems have been moved to a corresponding directory below the subsystem/ directory (for instance, the SMS Daemon can now be found as subsystem/smsd/).

### 12.3.3 Software build system

A build system has been partly established. For the Java subsystems, Ant was already used as the build software, but for the NAV package as a whole, we now use the GNU utilities **autoconf** and **make**.

When fully established, building and installing NAV from source code will be a simple, auto magic process. This process can be used as part of building a software package for different Linux distributions or UNIX variants. E.g. we want to create RPM packages of NAV for Mandrake Linux and Red Hat Linux.

The NAV v2 installer can simply be described as a large, monolithic script that knows everything about installing every little part of NAV. The new build system delegates the responsibility for installation to the developer of each subsystem. If the build and/or installation procedure of a subsystem changes, the developer who made the change must also change his Makefile.

At the root of the source code repository lies the `configure.in` file, which is input to the `autoconf` tool. `Autoconf` reads `configure.in` and creates a shell script called `configure`. This script will configure the software package "NAV" for a build on a particular platform. The script can be run with parameters to set the installation path of different parts of NAV. This enables us to build a NAV package to be installed somewhere different than the default `/usr/local/nav/`.

The `configure` script will search the local system for prerequisites needed to build the NAV package. Every subsystem will have one or several Makefile templates, called `Makefile.in`. These templates contain macros referring to the installation paths of NAV and the tools needed to build NAV. The templates are processed by the `configure` script, and the macros are expanded, resulting in pure Makefiles.

A Makefile is input to the `make` tool, which, generally speaking, is a tool to resolve dependencies between files. `make` is probably the most common build tool in any UNIX environment. To build the NAV package (that is, prepare the source tree in whatever manner needed to produce installable files - such as compiling source code into binaries), all one has to do is issue the `make` command in the root of the source code tree. When the package has been built, it can be installed by issuing the command `make install`.

If one wants to build or install only one of the subsystems, one can enter that subsystem's directory and issue one's `make`-commands there. This is very useful for a developer who wants to install only the subsystem he/she just made a bug fix to.

#### 12.3.4 Restructuring installation layout

All the code of NAV v2 was hard coded to find all NAV related files below `/usr/local/nav`. NAV v2 also featured the principle of splitting the installation into two mirroring directory hierarchies `navme/` and `local/`, where the former contained the installed version of NAV and the latter was meant for local extensions and any data that was generated by NAV (such as log files, Cricket statistics and so forth).

The build system enables us to configure NAV as a software package to be installed in any location in the file system. Different parts of NAV can even be installed in different locations. The goal is for a typical NAV installation to have a typical UNIX-like directory layout directly below `/usr/local/nav/` (or any other prefix) such as in figure 31.

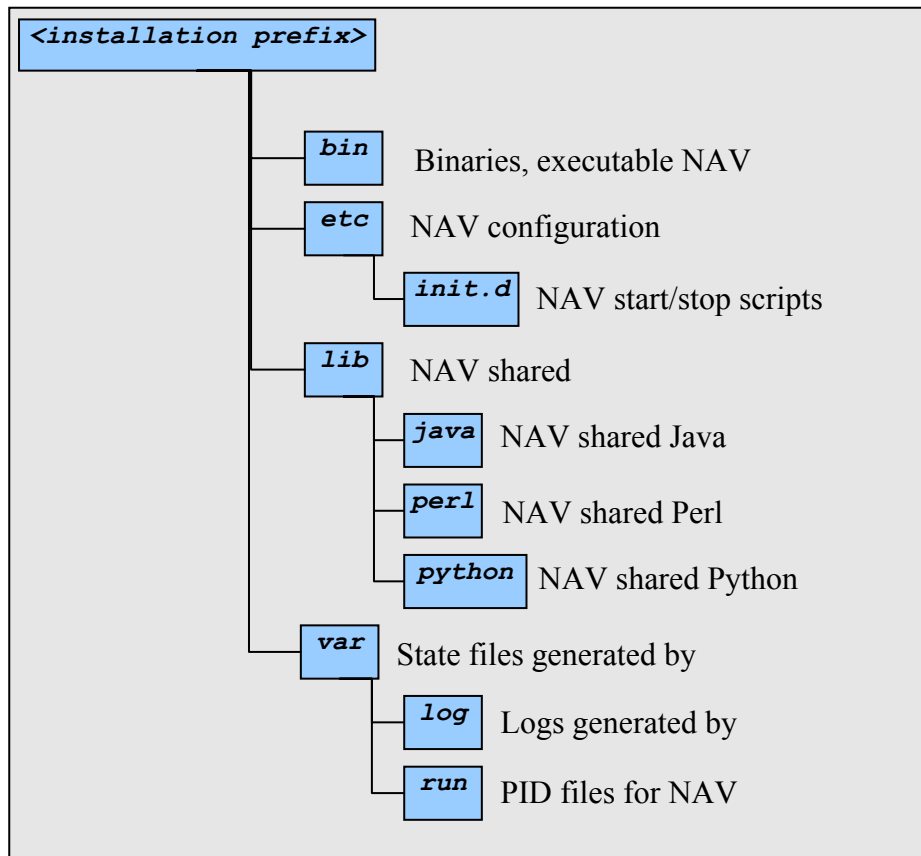


Figure 31: NAV v3 preferred installation layout

To achieve this, the build system dynamically creates library modules for both Perl and Python, containing the full paths to the different parts of NAV as configured by the configure script. A Perl or Python script that needs to know where a NAV related file resides needs only to include its respective path module.

Previously, these scripts also needed to hardcode the path to the actual library modules to include. This should no longer be necessary - it is now required that the NAV libraries are on the search path of the respective programming language. For Perl and Python this is typically achieved by respectively setting the `PERL5LIB` and `PYTHONPATH` system wide environment variables.

While many scripts have been converted to use these modules to discover the correct paths for NAV files, the hunt for scripts using hard coded paths is still going on. Also, this strategy has not been completed for Java programs.

## 12.4 Problems

No particular problems were encountered in this subproject.

## 12.5 Further work

- All hard coded NAV paths must be found and eliminated from the source code.
- A path module for Java must be generated by the build process.
- Makefile.inS must be written for every subsystem.
- There are still files that need to be moved within the Subversion repository, from the old location into the subdirectory of their respective subsystems.
- A system for generating packages (rpm, deb and so forth) for the most popular Linux distributions must be devised.
- Prerequisites of NAV that do not exist in packaged versions for these popular distributions should be packaged by us and offered in an ftp repository as a service to NAV users.

## 12.6 Concluding remarks

Unfortunately, this subproject wasn't very well specified in the original project plan. The activities have evolved quite a bit, and as a result, the actual number of hours used has grown beyond the budgeted hours.

Although the objectives weren't achieved fully, the activities of this subproject have laid a solid foundation for a flexible configuration and installation system for NAV 3.0. Work on these activities continues unabated after the conclusion of the tigaNAV project.

## 13. NAV version 3

### 13.1 Summary of NAV v3 features

tigaNAV introduces many new features to NAV. We have also elaborated on existing NAV v2 features, in some instances by replacing the code. Other features are ported without added functionality.

Table 32 on the next page gives an overall overview of the complete NAV v3 system. Some explanations and remarks are in order:

#### Legend

- Implemented prior to project NAVMore, 2001 or earlier.
- New or radically improved in project NAVMore (2002).
- ▲ Work done in project tigaNAV (2003).
- (▲) Not completed in tigaNAV, to be done in 2004.

#### Code maturity

Alpha Merely a proof of concept.  
Beta Functionality in place, but needs further testing.  
Stable Only minor bugs, if any.

#### The first letter of the chapter reference

- t The tigaNAV report (2003).
- o The NAVMore report (2002, in Norwegian).
- e The NAVMe report (2001, in Norwegian).

#### Remarks (regarding tigaNAV results)

- 1 Accurate modeling of modules, parallel IP address spaces, closed vlan.
- 2 Radically improved: OIDdb, plug-in-based, type classifier.
- 3 Text files aborted, web front end.
- 4 History of physical devices with milestone events.
- 5 Uses the OID database.
- 6 Supports reuse of VLANs.
- 7 Uses the OID database
- 8 Same header and footer for all tools, common menu navigation, use of templating in the code.
- 9 One user database for both web access and alert profiles.
- 10 Radically improved, flexible, introduces hasPrivilege.
- 11 Not only boxDown status, also other events.
- 12 Message system to inform NAV users of planned outage, errors or other operational events.
- 13 All information on one device in one place / web page.
- 14 Will be part of Device Browser.
- 15 Tree structured graphical display of the network topology on a per vlan basis.
- 16 New feature: all machines behind a switch port.
- 17 Combine many RRD data sources on the same web page, or even in the same graph.
- 18 Reports on outage of modules in a chassis or a stacked (physically or virtually) switch.
- 19 Flexible threshold monitor allowing threshold to vary depending on the RRD data at hand.
- 20 More thoroughly documented in the NAVMore report.
- 21 Very flexible and general solution.
- 22 Also services and entire rooms can be on maintenance.
- 23 From CVS to subversion.
- 24 Not monolithic, but modular.

# NAV status 2003-12-01

	v2	v3				Code maturity			chapter	remark
		ported	improved	replaced	new	alpha	beta	stable		
<b>Data collection</b>										
NAVdb - the network model	■		▲					✗	t5.5.1	1
Data collection system	■			▲			✗		t5.5.2	2
Seed files / web solution (editDB)	●			▲			✗		t6.3.1	3
Device management					▲	✗			t6.3.3	4
ARP-logger (IP - mac data)	●	▲						✗	e4.5.8	
CAM logger (mac - switch port data)	■		▲				✗		t5.5.3	5
Network topology discovery	●	▲					✗		t5.5.4	
Vlan discovery	●		▲				✗		t5.5.5	6
<b>Statistics</b>										
Cricket network statistics collection	●		▲					✗	t7.2.2	7
Cricket server statistics collection					■		✗		o3.3	
RRD roundtrip and packet loss					■			✗	t10	
<b>GUI and Users</b>										
Consistent web user interface					▲		✗		t3	8
User database	●			▲			✗		t2.3.1	9
User authorization	(●)			▲			✗		t2.3.2	10
User preferences				▲			✗		t3.3.5	
<b>Tools</b>										
Operational status page	●			▲				✗	t6.3.2	11
Message system					▲		✗		t8	12
Report generator	●	▲						✗	e4.5.1	
Switch port to room data					(▲)					
Device browser					▲		✗		t9	13
- Currently active switch port status					(▲)				t11.3.2	14
- Recently used switch ports					(▲)				t11.3.2	14
Network explorer					(▲)				t11.3.3	15
Network load map	●	(▲)							e.4.5.3	
Machine tracker	■		▲					✗	t11.3.1	16
Sorted statistics					(■)				o3.2	
RRD browser					▲	✗			t9.3.5	17
Allocated prefix matrix	■	▲						✗	o2.7	
Cisco syslog analysis	■	(▲)							o2.4	
NAVlog system	■	(▲)							o2.4	
<b>Monitors</b>										
Status monitor	●			■				✗	o3.4	
Module monitor					▲		✗		t5.5.2	18
Service monitor					■			✗	o3.5	
Threshold monitor	●			▲		✗			t7.2.5	19
<b>Events and Alarms</b>									t4	20
Event engine and event queue					■		✗		o3.6	
- shadow alarms	●			■			✗		o3.6.3	
Alert engine	●			■			✗		o3.8	
SMS alarms	●	▲					✗		t4.3.3	
User alert profiles	●			▲			✗		t4.3.4	21
Maintenance	●			▲		✗			t8.3.2	22
SNMP trap reception	●	▲				✗			e4.5.6	
<b>System</b>										
Software version control	●			▲				✗	t12.3.1	23
Software build system	●				▲		✗		t12.3.3	24

Table 32: NAV v3 features

## 13.2 NAV v3 Test and Release Plan

Our initial release plan was to have a running alpha test at NTNU by October 1 and have NAV v3 shipping by December 1. Due to delays (and we comment further on that in the next chapter) this has not been possible.

We have reviewed our estimated release schedule:

Activity	Timeframe / deadline
Alpha test	December 2003, January 2004
Beta test	February - March 2004
Development of (▲) - features	December 2003 – April 2004
Release NAV v3	April 2004
Wish list new features	Deadline May 14. 2004
Functional requirements v3.1	Deadline June 4, 2004
Development NAV v3.1	Summer 2004

This schedule may change. Consider it the recommendation of the tigaNAV project leader.

Further comments:

### Alpha phase

This stage focuses on testing of basic functionality. The testing will be formalized with a test plan specifying all the functional aspects that should be tested. The actual testing will be done by the NAV staff in ITEAs network group (3-4 persons). Tests that fail will go back to the developers for bug fixing. The testing will include setting up test network devices (testing module monitor, shadow tests etc). All tests must pass before we enter the beta phase.

### Beta phase

We would like three beta installations of NAV: NTNU, UiTø and HiMolde. The purpose of the beta phase is twofold:

1. Feedback from NAV users on bugs they find, feedback on the user interface, usability etc. Bugtracker will be used for bug reporting.
2. More intricate testing by the alpha test team, including special scenarios that might occur in a production environment (all the stupid things users may do, all the strange ways one may design a network ;)).

### Development of (▲)-features

As table 32 points out, not all planned NAV v3 functionality is implemented. The rows marked with the (▲) symbol need completion. There is also a need for improvements in other tools. Please note, we are *not* talking about *new* features, but completing planned NAV v3 functionality.



This work will be done with deadline of April 2, 2004, prior to the release of NAV v3.

New features for NAV v3.1

There will be a general feature freeze on NAV v3 until the summer of 2004. We also want the next development project to be less ambitious, not introducing fundamental changes to NAVdb, the data collection system or event / alert engine. The development will primarily focus on adding new functionality to the existing tools.

The next development project should also have a formalized requirement specification. The requirements should be based on NAV users' input and the NAV staff's overall thoughts. We have set a deadline for NAV user feedback giving enough time to write the requirement specification.

## 14. Summary and Conclusion

### 14.1 Project Self Criticism

Development of NAV v3 started with project NAVMore in the summer of 2002, one and a half year ago. We have continued with tigaNAV and we see now, finally, a complete system, at least in its alpha phase. Release is set for April 2004.

Ideally a software development project should not use so much time from implementation to rollout. In our defense we should add that some of the early NAV v3 results have been in use at NTNU. But still, the project leader certainly takes criticism on the overall progress.

There are many reasons for the delays:

- NAV has always been an ambitious project, NAV v3 has perhaps been *too* ambitious. For the event and alert system and for the data collection system we reached a point of no return. We saw that hours and progress were growing way beyond budget. Still the belief in the new and improved system has been strong. The project leader means, and has always meant, that it is worth it. Yes – we are delayed. Yes – planning has in some cases been bad. Yes – the project leader has received and *deserved* criticism on project leadership.
- In addition there has been the shift towards more professionalism in the software development process. The basic perl-scripts of the old days will no longer do. Neither will the impulse motivated way of programming. A paradox perhaps, because this has definitely been one of our strengths. We have seen a problem at hand, grasped it and solved it. An important aspect has been the tight couple between the engineers running the network and the developers. In some cases the network engineers themselves have done the development. Again – a benefit of being small, but it will not do anymore.
- The shift towards more professional software engineering has also had its downsides. This has been necessary, no doubt. And on a general basis for the highly qualified development team, it has been no problem. In fact it has been motivated by the team itself. But there are examples of personnel not being able to adopt to this new level. For them their era as NAV programmers may be over (but their contribution will not be forgotten!).
- Not only has NAV grown as a product, the development team has also grown. tigaNAV has had 14 persons in work – a large group to administrate. The personnel have also been from different groups within ITEA and two programmers have been from UNINETT. There have been some “cultural” challenges. The expanded focus on system

management has also, in some cases, introduced conflicts in goals, or at least conflicts in focus. We have also had examples of personnel being delayed, not delivering according to plan, not showing up on meetings and so forth. A stronger project leadership may have resolved some of these issues, but not all.

- Then there has been the mishap of altering the data collection system on the development machine, which in fact has resulted in poor test data for the tool developers. Our intention was to avoid this by using the old collection system, but since we (unfortunately!) introduced database changes at a far too late point in time, we reached a stage where data were inconsistent. This has definitely introduced frustration and delays for tool developers.
- Another underestimated challenge has been the training of new project members. For the project leader and the “base crew”, NAV is a part of life, so to speak. We have a lot of knowledge from our history, we take certain things for granted. We also have an understanding of the network we want to manage. For new personnel it must have been difficult to grasp it all. The project leader has not spent enough time to communicate our ideas and thoughts. And since documentation in some cases is poor, there have been few alternatives. Having said this, discussions on the nav-developers list have resolved many issues.
- The project plan has been too vague. Prior to development an extensive requirements specification should be made. This has been skipped due to lack of time. The “base crew” has had an overall idea of what we want, but for the developers a written list of requirements is far better. And in fact, all aspects have not been thoroughly thought through beforehand, resulting in fundamental changes at a far too late a stage in the project lifetime.
- Finally subsystem integration has been poor. In fact it has hardly been planned. The project plan focuses on each subproject and the work within it. The challenges with integration have been skipped. This should be a more structured process, and it should not be delayed to the very end. A suggestion for the future is to develop separate, but system-integrate weekly. A separate installation should be running at all times with nothing but working code from the versioning repository.

So it seems, we are not perfect? There is room for improvement. Still, having said that, on an overall basis, we should be pleased! A very enthusiastic group – a highly *competent* group – has worked hard (at times overly hard) and produced a lot of good stuff.

With all self criticism left behind, let us conclude in a more optimistic manner. Because, no doubt, speaking in technical terms, speaking of real world results, the tigaNAV crew has definitely made many great improvements:

## 14.2 Conclusion

The development of NAV has from its very beginning in 1999 till now been an iterative process; we have gradually evolved. NAVdb and the data collection system is the heart of our system; we have used a lot of energy to improve, and improve again, this fundamentally important base. tigaNAV has been no exception. We must emphasize however, that the achievements done now are final.

tigaNAV introduces great improvements with the OID database and a (semi-) automatic type classifier. Data collection is done in parallel within a plug-in based architecture. The NAV v2 seed files are history, the edit database tool gives a far better user interface.

tigaNAV has also focused on NAV as a software product. NAV v3 will be consistent in look-and-feel, we have adopted a general and powerful authentication and authorization mechanism, we have a proven version control system and we have introduced a standard programming language (python) in the user interface combined with use of templating. We have also replaced the monolithic installation scheme with a modular software build system.

Finally tigaNAV introduces many new features to NAV. Not everything is finished, but it will be within the timeframe set for NAV v3 release in April 2004. Table 32 in the previous chapter gives a good overview of all NAV v3 features. To summarize, the most significant tigaNAV *functional* results are:

- A more general operational status page with status on all operational events (eventually). An integrated message system to inform NAV users of scheduled outage, special faults and other operational events.
- The device browser presenting all information on a device in one web page with links to reports, statistics, switch port data etc.
- The network explorer introducing a graphical display of the network layout on a per vlan basis.
- An RRD browser with the ability to gather different statistics on the same page or even in the same graph.
- Device management with the ability to track milestone events of physical devices from order and arrival through the operational stages.

tigaNAV has certainly improved NAV in many aspects. The project has suggested a timeframe for NAV v3 release. We recommend a more conservative model for improvements in 2004. The overall goal must be to offer a stable NAV v3 for NTNU and for interested UNINETT members.

After that – after all – the NAV must go on!