NewProfiler - BSc Report

Martijn Bleeker           Ruben Wieman
March 20, 2008
Preface

This document describes our Bachelor project which we have done at the Delft University of Technology. The project is a requirement for our Bachelor of Science grade in Computer Science. Our supervisors are Ir. B.R. Sodoyer and Drs. P.R. van Nieuwenhuizen.

The application that has been developed is NewProfiler and the company for which we developed it is NewNomads[8]. NewNomads offers small to medium business software solutions such as websites, marketing and communication services.

The purpose of this project is to demonstrate our understanding and skills in developing a software application and it is one of the requirements for our Bachelor of Science (BSc) grade. It is also an opportunity for us to experience the process of developing full-scale software in the real world. Furthermore, the software should be of real use to NewNomads.

We want to thank Ir. B.R. Sodoyer and Drs. P.R. van Nieuwenhuizen for supervising our project. We also want to thank Michel and Floor from NewNomads for providing us with this project and for the help, inspiration and trust they gave us.

Delft, March 2008

Martijn Bleeker
Ruben Wieman
Abbreviations

This section contains a short description of the definitions of abbreviations used in this document. For the complete definitions, we recommend that you look them up in the dictionary or on the W3C website[5].

**BSc** Bachelor of Science - The bachelor degree of science.

**HTML** HyperText Markup Language - A set of standards used to tag the elements of a HyperText document.

**WSDL** Web Service Description Language - A XML format to describe the communications protocol for SOAP messages.

**XML** eXtensible Markup Language - Used to mark up documents that are used in communication.

**SOAP** Simple Object Access Protocol - A set of conventions used for invoking web services.

**UML** Unified Modeling Language - A language used to model and visualise a system under development.

**URL** Uniform Resource Locator - Protocol for specifying addresses on the network.

**SQL** Structured Query Language - Language for database communication.

**IP** Internet Protocol - Protocol for spitting and addressing packets of data.

**ID** Identification - Identifier.
## Contents

1 Introduction ................................................. 6
    1.1 Outline of the report ............................... 6

2 Planning ..................................................... 8
    2.1 Teamwork and individual work ....................... 8

3 Analysis and Requirements .................................. 9
    3.1 Analysis of the project and requirements .......... 9
    3.1.1 Functional requirements .......................... 9
    3.1.2 Non-functional requirements ...................... 9

4 Design ........................................................ 10
    4.1 Design layout ....................................... 10
    4.1.1 Package layout ................................... 10
    4.1.2 Module overview .................................. 11
    4.1.3 SOAP Elements .................................... 11
    4.1.4 Cache ............................................. 11
    4.1.5 Logger ............................................. 12
    4.1.6 Filter .............................................. 12
    4.1.7 Report and Home report .......................... 12
    4.1.8 SiteMap and SiteMapManager ...................... 13
    4.1.9 Contact and Order ................................ 15
    4.1.10 Mailing .......................................... 15
    4.1.11 Core .............................................. 16
    4.1.12 Customization .................................... 16
    4.2 Modifications to the design ......................... 16

5 Implementation ............................................... 18
    5.1 Approach ............................................. 18
    5.2 Problems and solutions ................................ 18

6 Testing ....................................................... 22
    6.1 Cache ................................................ 22
    6.2 Filter ............................................... 22
    6.3 Core ................................................. 23
    6.4 Contact .............................................. 23
    6.5 SiteMap ................................................ 23
    6.6 ContactGroup ........................................ 24
    6.7 Mailing ............................................... 24
    6.8 Report ................................................. 24
7 Results
7.1 Overview of project result ........................................... 27
7.1.1 Features implemented ........................................... 27
7.1.2 Features not implemented ....................................... 28
7.2 Timeline ................................................................. 28
7.3 Reflection ............................................................... 29
7.3.1 Planning ............................................................. 29
7.3.2 Analysis .............................................................. 30
7.3.3 Design ............................................................... 30
7.3.4 Implementation .................................................... 30
7.3.5 Testing .............................................................. 31
7.3.6 Communication .................................................... 31
7.3.7 Project management ................................................. 32
8 Conclusion ................................................................. 33
9 Recommendations ....................................................... 34
9.1 Incomplete tasks ....................................................... 34
9.2 Problems and solutions .............................................. 34
9.3 Additions to NewProfiler and the Desktop ....................... 35
A Installation .............................................................. 38
B Project description ....................................................... 38
C Analysis document ...................................................... 38
D Design document ........................................................ 38
1 Introduction

The internet of today expands at an incredible rate. More and more people surf the web every day, buy products online, gather information or just seek entertainment. Almost every company has a website to address a growing market. For many companies, a website is at least as important as a physical location to display their products. For many small companies however, creating a good and attractive website is too expensive or simply too difficult to maintain. NewNomads addresses this problem by providing impressive websites, hosting and maintenance. Also marketing services are offered to improve the effectiveness of the website. Today's marketing usually consists of mailing (potential) customers using leaflets, e-mail campaigns or other advertisements. However, this approach can be somewhat inefficient as it effectively addresses everyone with the same information and it does not provide a good feedback to the sender, except for statistics that can be created from the total company turnover. Many people get advertisements and offers about products they are not interested in and company owners can only indirectly draw conclusions from their marketing campaigns. It is obvious that there is a lot of improvement to be achieved in this whole marketing system.

That is where NewProfiler fits in. NewProfiler is a behavioral target market system that offers many improvements to the marketing system as described earlier. NewProfiler is a tool for sending out e-mailing campaigns to targeted groups of contacts based on their profile. It measures response to those campaigns and provides the user with statistics about activity on their website. This system is integrated with other services provided by NewNomads and together with these services it provides a complete solution to all the needs for web-based marketing and sales.

1.1 Outline of the report

This report describes how we developed the NewProfiler application. It starts with the project planning in chapter 2. Here we show the original timeline for the project and there's a description of how we divided the work. After that we focus on how the project analysis was done and how we gathered the project requirements (Chapter 3). It gives a short summary of the key requirements for this project. We describe how the system was designed in chapter 4. Each of the chapters in the design document has a reason why we included it. Also this chapter explains what changes were made to the design in later stages of the project. The implementation chapter (Chapter 5) contains an explanation of the problems we encountered while building the system. During the implementation and continuing after completion, we tested the components. This process is explained in chapter 6. We also discuss how much cases the tests have covered and what parts might need attention after this project has been completed. Some problems and bugs are discussed to illustrate the things we solved during this project. The results of this project are presented in chapter 7. We discuss how long the various phases of the project actually took and why things did not go as planned. We analyze what was learned from this project and what we should do better next time. Chapter 8 contains a summary of the project
as discussed in this report and the conclusions we can draw from our work. Finally, we present a number of items that should be addressed after this project and offer a list of ideas and recommendations for further development in chapter 9. In the appendices we have included the original project description, the analysis and the design.
2 Planning

The original planning promised to create the complete system within a time span of four months. We soon discovered that this planning was not realistic and that we could not complete the project within the planned amount of time. We explain later why we did not succeed in completing the project within the time as displayed below.

<table>
<thead>
<tr>
<th>Id</th>
<th>Taaknaam</th>
<th>Begindatum</th>
<th>Einddatum</th>
<th>Duur</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Analysis</td>
<td>4/16/2007</td>
<td>4/27/2007</td>
<td>2w</td>
</tr>
<tr>
<td>2</td>
<td>Design</td>
<td>5/7/2007</td>
<td>5/25/2007</td>
<td>3w</td>
</tr>
<tr>
<td>5</td>
<td>Report</td>
<td>7/9/2007</td>
<td>7/20/2007</td>
<td>2w</td>
</tr>
<tr>
<td>6</td>
<td>Presentation</td>
<td>8/6/2007</td>
<td>8/10/2007</td>
<td>1w</td>
</tr>
<tr>
<td>7</td>
<td>Exams</td>
<td>6/18/2007</td>
<td>7/6/2007</td>
<td>3w</td>
</tr>
</tbody>
</table>

Figure 1: The Gantt chart from the original planning.

This BSc project should be completed within one quarter of the academic year. It covers the complete path of building a software system and is eventually completed with a presentation, presenting our results and experience.

Our first planning turned out to be too tight and the project too large to complete within the stated period of time. We discuss this in section 7.2.

2.1 Teamwork and individual work

The analysis phase of the project was completely done by the two of us. There was no need to split the work. In fact, doing this phase together has helped us correct things that one of us might have overseen.

During the design phase we worked together for most of the time. Although we split the design for the Cache and the Filter. These were two big parts of the project. The rough design was done by both of us, but filling in the details was individual work.

In the implementation phase we decided to split up the work. The Filter and Cache were implemented by the person that designed it. Also this was done first, because both are essential components of the project. Without these, implementation of the modules would have been at least very difficult. The remaining modules were divided between us.

During testing we basically kept the same distribution of modules. We redivided some modules because testing of some parts proved to be very time consuming.
3 Analysis and Requirements

3.1 Analysis of the project and requirements

We started the analysis with a design document draft, made by the contractor. This draft wasn’t complete, but it was a nice base to start with. After our first draft of the analysis report was done, we had a meeting with the contractor to discuss some features and to see if we left imported things out.

NewProfiler had to be an application which could run on the existing NewNomads Desktop. This is a web-based platform which looks like a local computer desktop. Since the NewNomads Desktop uses tomcat to provide it’s services to the outside world, NewProfiler also had to be written in Java, as a module of the NewNomads Desktop.

3.1.1 Functional requirements

As usual, the first ideas on how to create a particular program are not always the best ideas. This section focusses on what changed in the requirements during the project. The complete list of requirements is in Appendix C

- **NewProfiler home** - This part of the project was supposed to be the welcoming screen for visitors of the NewProfiler site. Some statistics would have been shown about the web shop it was running on. We decided to drop this part because the Report module covers the same functionality. If demand rises for an opening screen with statistics, the Report module will be able to provide that functionality.

- **Conversation module** - The conversation module has been dropped from the project, though it’s functions still exist in NewProfiler. We decided to put the methods of the conversation module into the contact module. We thought this was a logical choice, since the conversation section depends on the contact module and it’s functions.

- **Order module** - The order module is not part of the generic project anymore. Not all websites managed by NewProfiler will handle orders and for the ones that do, a generic implementation is practically impossible to design.

3.1.2 Non-functional requirements

- NewProfiler should be very scalable. When the logger has gathered a lot of data, it shouldn’t affect performance too much.

- Speed is very important, a request shouldn’t take more than seconds to process.

- The system is supposed to be very robust. A longer uptime shouldn’t have impact on stability. Also there are extensive ways to apply user roles. This is functionality that is inherited from the NewDesktop platform. This limits the probability that unauthorized users can negatively affect the system.
4 Design

4.1 Design layout

Each section in this chapter refers to one or more chapters in the design document. The sections explain what design problems we discussed during the design phase and how the design can be used to implement and perform maintenance on the system.

The design document contains all of the design decisions we made for this project, it contains tables with database fields, SOAP elements and a lot of other material that is usable for later reference. It provides guidelines for the implementation of the various components. The layout of the system is explained using UML. An important decision we made was that we would not design all of the methods and variables in detail. The design document contains mostly input definitions, output definitions and a global layout and strategy of the system. The actual implementation is left open to the programmer. This approach suggests that a lot of documentation must be placed in the code itself. The code will explain itself but the JavaDoc inside the code will be a useful help to summarize the effect and purpose of the code. If later maintenance is required on the system, reading the design document will not suffice. The developer will also need to examine the code in order to perform maintenance.

The design document is written in English because we use many English definitions of protocols as well as English names of services and systems. It is both confusing and annoying to mix English with Dutch so we decided to do the entire document in English. Furthermore, not all of NewNomads’ employees have Dutch as their native language. We want the documents to be usable by everyone.

4.1.1 Package layout

The software package name is about as important as the name of a source file. It describes the owner (nl.newnomads), the project (.newprofiler) and the component of the project (.module, .servlet, .cache, ...). With only a few words you know exactly where you are and from what part of the project you use subroutines. It is of great importance to name the packages and source files properly to prevent confusion and namespace conflicts. The ‘NewProfiler package’ chapter in the design document provides an overview of the packages and can be used as a programmer reference.

The common prefix for most of the NewNomads Java source is nl.newnomads.newprofiler. We did not work outside this package thus we did not make any modifications to other projects such as Desktop (nl.newnomads.core) package, which is the framework on which we built NewProfiler. We did have access to the source and documentation of other packages. The Desktop framework provided us with a good example and a lot of information on how to create the NewProfiler modules as well as on how to integrate with the Desktop. NewProfiler uses a similar layout as other components of the Desktop framework.
4.1.2 Module overview

The module overview contains all of the modules in NewProfiler. The ‘Module overview’ chapter provides an overview of the modules, the method names and a short description of what a method does. Each module has its own section further in the document. The difference between a module and a non-module system component is that a module is actually an extension of the abstract ‘Module’ class in the Desktop framework. The modules are components that can be configured and loaded into the framework. SOAP requests can be relayed to those modules by the Desktop framework. Non-module components such as the Cache and the Filter are components that provide miscellaneous services or can be used as data container. They are sub-programs which can be loaded by a module.

The ‘Module overview’ chapter is often used to look up a method name. Each of the module sections contain the database design for that module, a services overview and a complete definition of the SOAP requests and responses.

4.1.3 SOAP Elements

The SOAP elements definition is an overview of all the SOAP elements used in the system. We used it a lot during the implementation of the modules, during construction of the configuration XML and during tests. It will be useful whenever work needs to be done on any of the SOAP components. Each element describes all of the sub elements it could possibly contain. Still, you have to look up the method details to see what elements are actually required or returned. An alternative way of looking up these details is to examine the config.xml that comes with the NewNomads Desktop. This configuration file contains a method-by-method description of the input and output components.

4.1.4 Cache

The Cache section contains the design decisions of the cache. It contains the design of the Cache and subcomponents such as the objects in cache and the SiteMap management component. We used this section a lot during the implementation and it will be of further use when maintenance is required on the cache.

The reason why we introduced a local cache into the system is because we compared the scalability and speed requirements to that of another NewNomads project. This example project is called ‘SimpleReporter’[7]. It can best be described as a combined contact, order and report module all in one small system. The program reads the database and generates statistics and reports from server activity. However, this system is very slow and it already has succumbed a few times to the amount of data in the database. The problem with this system is that it does not use any optimizations for processing the data. The cache in NewProfiler is one of the optimizations that target this problem. NewProfiler must be able to handle much more data at an even faster rate. We used Introduction to the Theory of Computation[3] to identify the problems. Fundamentals of Database Systems[2] contains much information about caching and database systems. We used both books to design the database and the cache.
The cache supports multiple running modes. You can choose between speed and safety. The speed option greatly reduces the number of queries done on the database. The problem with database communication is that initializing the communication usually takes more time than executing the task. We designed this mode to support scalability of the system. With a lot of users working simultaneously, there can be a lot of wait time due to communication and database locks. The cache is designed for asynchronous access to local data and will greatly speed up database operations.

The safety mode is recommended for development and running in unstable environments. Any database manipulation calls are forwarded to the database immediately. This will ensure that any updates are made permanent immediately and do not get flushed once in a while. Items in the database are still cached locally so they can be read quickly.

### 4.1.5 Logger

The Logger section contains a step-by-step script for logging a request. It contains all of the information needed to implement and we used this section a lot during the implementation phase. Because the logger does not offer any additional services, this section does not contain any useful information except for performing maintenance on the logger.

### 4.1.6 Filter

The Filter section describes the component layout of the filter, the query syntax and explains how the query is handled internally. This section has many uses:

- We used it to implement the Filter. It contains both the UML design and a description of each component.

- This section also defines the query syntax. When implementing the client software, it is necessary to examine the query syntax in order to deliver a proper implementation.

- The FilterElements are explained here. When extending the filter, this information will be of much use.

The module associated with the Filter is the TargetGroup module (which was later renamed to ContactGroup). This module allows storing filter settings in the database. (e.g. query, includes, excludes) They can be reused later. It also allows NewProfiler to cache the query and keep a parsed version in memory. There was little discussion about this module and the design only contains the specification of the SOAP methods. It is used as an implementation reference only.

### 4.1.7 Report and Home report

NewProfiler should generate a report in two different ways: The first is on the ‘Dashboard’. That is the first page when the user starts the program. On this page, the last 24 hours of activity are displayed. Next, the normal report shows the activity from any moment
in time, any selection of contacts etc. We designed two different report modules for these
tasks. The ‘Home report’ would display the overview of the last 24 hours while the standard
‘Report’ would output report information based on user selections.

The main difference with the standard report is that the Home report module generates
a ready-to-use report while the standard report generates an output that should still be
parsed by the client. The advantage of this approach is that generating the output would
be faster and with less communication overhead on the home page. When an extensive
report is required or the user needs to view additional data, the standard report module
will take over from the more simple home report.

We had a lot of discussion about the division between Report and Home report. The
Home report might be faster but we had no proof that it actually would be. Furthermore,
the problem with the division is that we would end up with two modules that would
duplicate some of the logic which is not a good programming solution. We decided that
these modules would somehow share their logic so this problem would be reduced to a
minimum. The discussion took a lot of time and we eventually yielded to the solution to
split these modules.

4.1.8 SiteMap and SiteMapManager
The SiteMap is a large module with a lot of different tasks. It provides services to manage
files in the public web folders, virtual resources such as server side scripts and it manages
virtual folders which can be used to group resources. There are many ways in which this
functionality can be implemented, so that offers a lot of material to discuss. Some of the
items in this discussion are summarized below.

- The resources also contain the type of resource (e.g. the Mime type). This inform-
  ation can be used to create folders based on resource types and automatically put
  resources in the appropriate folder. The server can automatically create virtual fold-
  ers for each resource type and handle exactly the same as folders created by the user.
  However, there are some problems that arise. First of all, the server needs to keep
  track of changes to the resources. Any changes need to be reflected on the placement
  of the resources. Next, the user can change the folders created by the server, causing
  the server to lose control over those folders causing problems to the automatic order-
  ing of resources. If we want to have this functionality in the future, a separate system
  needs to be implemented for these folders. Folders that are automatically generated
  would not be treated as normal folders. The user would not be able to change them.
  This solution was rejected because making an alternative implementation for special
cases is generally not recommended. Our solution was not to create any folders for
the different resource types. Instead, the client software should create folders for the
resource types upon requesting a resources overview. This solution allows the client
software to determine whether or not to display folders based on resource types. Fur-
thermore, the server has to do less work and some processing is required by the client
instead. This should be faster because less communication is required.
The SiteMap module can scan the server for new or missing files. Any changes to non-virtual resources can be automatically found and applied to the SiteMap layout. This feature requires some information in order to work properly. First of all, it needs a place where to start scanning directories. One solution is to start in the server root, try to find the web server directories and list files within those directories. The problem is then how to recognize the web server directories and find the proper domain directories, what URL to assign to the resources found and many similar problems. In general, the SiteMap needs to know some things about the server configuration such as virtual domains, accessible directories and the base URL of each directory. It is a lot of work to have the server find out this automatically so we chose an alternative solution. The user should tell the SiteMap everything it needs to know to start scanning. This solves all the problems about required information. The question is now how to provide the server with this information. One solution is to have the client include this information in the SOAP request for scanning the server. However, this would allow the user to mess up the resources structure if incorrect information was supplied. Furthermore, it requires the user to know the structure of the server. This information is generally not available to the user of the system, unless you are the administrator who installed the system. It is not much of a problem to restrict the use of this feature to ‘root’ accounts only but that defeats the whole idea of the feature. Any user should be able to update the database after uploading new content to the server. Thus this feature has to be safe and easy to use. We chose to hard-code all of the required information into the module. Hard-coding is necessary until the Desktop framework offers a way to import information from the configuration file into the module. Until then\(^1\), it is required to update the information in the code and recompile.

The SiteMap module has a lot of responsibilities in managing the folders and resources. It is easy to create a bad folder structure since a lot of problems can occur with folder and resource nesting. Missing references, cyclic nesting or unreachable locations are just a few of those problems. The Cache only manages the contents of the database, and in a few cases the relations between objects. Since the SiteMap is a lot about item relations, we did not want to put that responsibility in the cache. The SiteMap module already manages the SOAP connections and the mapping to the cache, so we did not want to add this responsibility there. Instead we chose to create an extension to the Cache. This extension called the SiteMapManager serves as a bridge between the SiteMap module and the Cache. It only manages the relations of all the items in the SiteMap while leaving the actual database management to the Cache and the implementation of services to the SiteMap module.

The resources can be put into folders to provide some overview and structure to the contents of the server. We discussed what restrictions to put on this system.

\(^1\text{At the time of writing, this feature is available.}\)
- Folders can either have a null parent folder (which is the root) or another folder as parent. If the same folder should be used at multiple places, a copy should be created for each location.

- Resources do not need to be put in a folder. Resources without parent folder exist in the root of the Sitemap.

- Resources can have multiple parent folders. The resource is displayed at each parent. For example, an image of a product could be put in both a ‘products’ folder and an ‘images’ folder.

4.1.9 Contact and Order

The Contact and Order modules provide services for connection to an external database such as a web shop database. These modules are some of the few modules that do not use the cache for their services. The design of these modules is very basic. The modules simply convert the SOAP requests to SQL queries and vice versa. We had little discussion about these modules.

The Contact module also contains any of the subroutines for conversations. We put the conversation handling inside this module because the conversations require the contact subroutines. Furthermore, if no contacts are initialized (e.g. the module is not mounted because the customer does not need it), conversations are also not needed. The Contact module can be split anytime into a Contact and a Conversation module. This might be better in the future because it creates more specialized and smaller modules. The Desktop framework now also offers better communication between modules. This communication problem was one of our main reasons for placing conversations and contacts in the same module.

The Contact and Order module must be updated for each customer. They must be changed to support the layout of the contacts database and the type of products. Because the Desktop framework did not have a good system for importing data to customize the module or to adapt components at runtime, the solution to this problem was left open in the design. The current implementation of the Contact and the Order module can be customized by copying them to the customer’s source folder. Use them as a template to complete the customer-specific implementation. When building the project, the template is automatically overwritten with the customized version.

4.1.10 Mailing

The Mailing module is one of the few modules that does more work than just communication between client and server and delegating tasks to the cache or database. It can keep sending e-mail messages even after the user has disconnected. This is the most important feature of this module. The user does not have to wait for it’s operations to complete. Thus, this module does not need to rush completing this operation. Sending too many messages at once is never a good practice since this could overload the mail channel.
4.1.11 Core

The last module is the Core. It is the first module to start in NewProfiler because it provides some services that other modules might require on startup. We designed this module as the central gateway for other modules. It starts the Cache and the FilterManager which is required by many of the modules. It also provides services that do not belong to any of the other modules. We designed an error-recovery feature that any privileged user could use to recover after a system crash or if the database was corrupt. This feature would only be implemented if we had some time left as it was not a required feature.

4.1.12 Customization

The Customization section contains a list of tasks that are required for setting up a new distribution of NewProfiler. This section was kept very short at first and we would update it as more information would become available during the implementation phase.

4.2 Modifications to the design

During the phases after the design, many changes were made to the design document as more information became available, problems arose and we became more experienced. The design document is kept up to date and we do strongly recommend to keep it up to date after this project. It contains many useful tables and definitions of system components as well as a lot of miscellaneous information about the system. The most important changes that were made are explained below.

- The Home report module has been discarded. All of the report functions are moved into the Report module. This module is also able to generate reports about a specific time window such as the past 24 hours. This module has been completely redesigned because the original design contained too little information about either the Home report or the Report module. Too many implementation details were left open and during the implementation we were unable to create a working Report module from the designs. The new Report module contains all the required methods for generating a report on the client side.

- Many naming errors are fixed. Mostly components of the SiteMap and the Target-Group had conflicting or unclear names. The tasks for the SiteMapManager and the SiteMap have been updated to prevent confusion. Other naming problems such as ContactGroup, TargetGroup and various cached items are resolved. Whenever we were confusing names during the implementation phase, we discussed whether it was appropriate to change the names in the design. In the current design, no different classes with the same name exist.

- A few small modifications to the database design were made. Some fields proved to be too small or incompatible with newer MySQL implementations. For example,
the IP address field in the Request table was only 15 characters long. This caused problems when the Logger tried to insert an IPv6 address. The design document did not mention anything about IP address length and during implementation IPv4 addresses were assumed to be used. Another change that was made is that the targetgroup, targetgroupinclude and targetgroupexclude tables have been merged. Updating three tables for a single internal object proved to be too cumbersome. The contents of the include and exclude tables can be stored in two fields in the targetgroup table. This greatly reduces database communication and overhead in storing the data. It has no consequences for the ContactGroup module because the Cache handles this change internally.

• Problem handling was removed from the Core module. The routines for problem handling proved to be a lot of work and there would be a great risk of security leaks. To make this feature fool-proof, user friendly and actually useful, a lot of more design work needs to be done, analysis of possible problems in the system and effective solutions that the system knows about. This feature is not a requirement for the system and it is too much work to be done within this project. Furthermore, if such a problem handling system is ever to be included in the system, it is much better to include it in the Desktop framework so it can target a much greater range of problems.

• The chapters about the JavaScript client program were removed. Instead, a Flash client will be implemented. This document is now only about the server side of the project and it describes the interface that can be used to communicate with the server. The client will later be redesigned by NewNomads’ Flash experts.

• Removing the JavaScript client from the design also affects the chapter about testing. We can no longer perform tests using this client. Instead, we will create internal test scripts that simulate SOAP calls on the system.

• Searching for resources (using the ‘searchFilesystem’ method) no longer removes missing resources from the database. Removing those resources can remove valuable information from the database such as requests done. The user might want to view this information, even if the resource no longer exists.

• The side-effects of removing folders from the system is now explained in the design.

• The customization chapter no longer exists in the design. Instead, it is a separate installation document. (See Appendix A).

• The old design document mentioned that a WSDL file should be created. This file is now generated automatically by the Desktop.
5 Implementation

5.1 Approach

The design of the system was mostly done together. We both had a good overview of the entire system and we had a good idea of the various components of the system. We decided to split the work so both of us could build the components in parallel. Most of the design discussions were done by now so we could start building the components without further intervention of discussions or missing design information.

Our approach was to start developing the two core components first. The components ‘Cache’ and ‘Filter’ are requirements for almost all of the other parts of the system. Then, these components could be tested so we had some certainty that the base of the system was working correctly. Finally, the modules must be built. The modules heavily depend on the two core components. In a few cases, methods of the cache were not yet available. These methods were replaced by stub methods, so the entire project could at least compile.

5.2 Problems and solutions

The problems we encountered were mostly related to design problems that were overlooked in the design phase.

Naming  During the design phase we came up with useful names for the classes in the system. We did not notice that some of the names were used twice in the design. For example, the SiteMap manages resources and folders in the system. We designed a SiteMap module that allowed the client to request the resources and folders from the server and we designed a SiteMap component inside the cache. This component manages the layout of resources and folders internally as well as queries on those resources. Both SiteMap classes exist in a different package but since they need each other, they have to import their counterpart. This quickly becomes confusing as you cannot do statements like SiteMap.staticMethod() because Java complains about ambiguous references. Of course it is not so difficult to tell the compiler which of the two classes you want to use but it is much more confusing to the programmer and to the reader of the documents. To prevent confusion, we renamed the SiteMap inside the cache to SiteMapManager. This new name is more fitting as this object actually manages the structure of the site map. The design document was updated and the naming was updated in the code. Next time, we will have to ensure proper names are used from the start, even cross-package.

A similar problem was that of the TargetGroup. The TargetGroup module allows the client to retrieve stored groups of contacts from the server. A filter query can be saved, allowing re-use and quick referencing of that query. However, the class that is used to cache the query is also called TargetGroup. Our solution was to rename the module to ContactGroup. The cached object is still a TargetGroup and all module method names refer to a TargetGroup. The client program will only see TargetGroups but the programmer can be confused by ContactGroup and TargetGroup. This is not much of a problem and
you quickly get used to it. The name ContactGroup is almost never used, except in the system configuration file.

**Bad values** The design document does not describe in detail who’s responsibility it is to check for bad values in the system. Data propagates through many different levels of the system and input can take place during various phases at runtime. In general, the client can only input data using SOAP messages. The NewNomads Desktop does only perform a few simple checks, such as proper XML syntax and illegal characters such as HTML characters, but there are plenty of faulty inputs that could make it into the system. It makes sense to check the values when the SOAP message is delivered to the module. However, this leaves open a few backdoors into the system. First of all, the logger component can insert data into the system. The logger does not use SOAP, it uses method calls instead. Since the logger does not do any verification by itself (it is considered a front-end component, it cannot be trusted), the server has to check input on those methods which are reachable from outside. Furthermore, the system can request an instance of a module and delegate operations internally (such as the Report module which uses the Contact module to get information about contacts). Thus the modules also have to check input on any of the services offered internally. Since these services are also used by the SOAP layer, we should be careful not to double-check the input. The Cache is used by many different components. It has a lot of responsibilities in the system and therefore it should be reliable and fast. It stores data in local objects and those objects can be accessed throughout the system. The problem is that the Cache should check as little as possible. The Cache does not know rules like ‘A password is required to be 4 to 16 hexadecimal characters’, it only knows what values can exist in what fields in the database. Thus it makes sense not to check variables for valid data except where it threatens database consistency.

There is another backdoor into the system which is only open during a small timeframe: The init step. During initialization, the Cache reads the entire database and constructs local objects. It is possible that the database is not consistent, for example if the server has crashed, a bug in the system has corrupted data or the user has changed the database manually. The Cache will also have to do some checks during this phase. Since the values in the database can never exceed the values allowed in the tables, some checks can be omitted.

We took all of these problems into consideration and came up with the following solution:

- Where possible, the module internal service methods (which are available also to other modules) must check for missing or bad ID’s where possible and other raw data such as Strings, numbers etc. They must also check for missing objects (null values) where necessary. There are a few cases where it is either safe not to check or not possible to check those values.

- The module internal service methods are responsible for providing the cache with additional synchronization hints such as performing flushes outside the usual schedule.
• The modules must check SOAP messages for bad values such as illegal characters and missing fields.

• The cache will only accept values that are safe to put in the database. It should reject any changes that would put the system in an inconsistent state. The values are stored in simple container objects (the actual cached objects). Those objects don’t perform any additional checks except for their cache state (such as not being allowed to update an object that is already marked for removal).

• A cached object is responsible for outputting valid data. For example, there are cases where an empty String ("") or a null-pointer have the same meaning. Since null-pointers can cause crashes or rejected database queries, the cached object will convert a null-pointer to an empty string. The cached objects are not allowed to change values but they are allowed to perform optimizations to improve speed or stability. Since the Cache is the only component that is allowed to input data into a cached object, the cached objects do not have to check their input data.

• The SiteMapManager is a component that manages the internal organization of resources and alike. The Cache will consult the SiteMapManager about any changes to the structure before updating the actual values of objects. Any bad updates will get rejected this way before any values are changed.

The Cache is the Achilles heel of the system. Almost all database traffic passes through the cache, the application performance is roughly defined by the speed and scalability of the cache and the entire program might crash if the cache does not work properly. We spent a lot of time optimizing the cache, double-checking for value ranges and ensuring that the checks are done at the correct locations.

**Thread concurrency**  
Thread concurrency is one of the greatest risks in the system. Incoming calls are done asynchronously. This is how servlets work. The Desktop framework can be considered to be one big servlet. Because a lot of information is shared between threads, a lot of problems can occur when threads interleave. Thread concurrency is less of an issue with modules working with multiple threads. The methods that handle requests only need to ensure that no global information in the module is changed. Only constants should be used, variables should be managed by the cache or the Desktop framework. The database connection pool ensures that no database query conflicts can occur because it locks connections when a module is working with it. There are a few exceptions where this is not possible and those exceptions are well documented by the connection pool. The modules only need to ensure that those routines are synchronized.

The biggest risk of thread concurrency exists in the Cache. The system can add, update, remove and read components from the cache simultaneously. We have put a lot of effort into ensuring that no thread concurrency problems occur in the cache. We use as few synchronization as possible as it slows down the system. There are few places where locking occurs. The cache uses lists that are safe to manipulate with multiple threads.
and the objects in cache keep track of their synchronization state. There are two modes in which the cache can do its work. An asynchronous mode and a synchronous mode. The asynchronous mode is faster but more risky when it comes to thread concurrency problems, since write actions are also done asynchronous. The synchronous mode can also have thread concurrency problems, but they are less common since write operations are synchronized.

**Features not available at project start** Parallel to this project, development of the Desktop framework also occurs. A few interesting features which were not available at the start of this project have become available during later stages. In the initial design, it was not yet possible to share information and services between modules. This feature has been implemented by NewNomads during our project. It is required for NewProfiler to share information between modules such as the Cache. The modules can make use of services provided by other modules, even modules that do not belong to NewProfiler.

Because the Desktop configuration file was very simple, it was not possible for us to import information from outside the system except from the database. Many variables inside the modules were hardcoded such as file system paths, the domain on which to operate, fields in the external tables etc. During our testing stage, a new configuration system for the Desktop has been implemented. The Desktop now uses a XML configuration file which can also contain information that is passed to the modules at startup. Our modules have not yet been updated to accept this information but they will be soon. This feature greatly reduces the need to recompile after changing variables or moving the system installation.

In the new configuration file, the required input and output of modules can also be defined. We have updated the configuration XML to reflect all of the input and output as specified in the design document. This also enables the Desktop to generate a Web Service Description Language file (.WSDL) that describes the interface for NewProfiler. This file will be of much use when constructing the client. This feature was not available when the system was designed.
6 Testing

The original plan for testing the server side of NewProfiler was to use the NewProfiler client for calling the methods and passing the arguments. The client application should test all of the methods available and print their results on the screen as though the user had requested the page. This plan was made while we still had the intention to develop a JavaScript client. Later on plans changed and instead of using a JavaScript client, the project will get a new Flash-based client. This client is yet being developed by colleagues from NewNomads and is not ready for use. This made us decide to let the server perform a selftest which emulates a client session and creates the necessary SOAP messages. Because there is no client for displaying the test results, we also let the server generate HTML pages with the output from the selftest.

6.1 Cache

This is one of the exceptions in NewProfiler with regard to testing. The Cache has been tested during development, the Cache is a very complicated part of the project and it would have been unwise to implement everything without partly testing its workings. All cache items have been tested during the test of their corresponding modules.

To test the Cache, we developed a program that simulated calls from outside. The program did thousands of asynchronous calls within a few seconds, then verified the contents of the cache with that of the database, and matched both against what it expected to see.

Running the test for the cache did uncover a problem that we weren't able to solve yet. The Cache can run in two different modes. One is the synchronous mode, the other the asynchronous mode. The asynchronous mode is a lot faster than the synchronous mode. This is something we saw while testing and matches our expectations. However the asynchronous mode contains a bug we were not able to locate, that prevents us from using this mode. After a batch of database/cache transactions the contents of the cache and database are slightly different. This is something that is unacceptable, but fixing this bug would cost us time we do not have during this project. In the future we will fix it, but this will happen when NewProfiler is actually being used by customers and the extra performance is needed.

6.2 Filter

The Filter has also been tested during its implementation. This is because of the same reason we tested the Cache during development. The Filter is too complicated to use big-bang-testing. Every component of the Filter has been tested right after completing implementation. First up was the Parser, which, after a few minor glitches, provided us with the output we expected after feeding it a query string. The second filter test was for the FilterElements. These are the classes that do the actual filtering. Despite all FilterElements inheriting methods from one generic FilterElement, they were each tested individually to ensure correct functioning. With exception of some minor bugs in the
generic FilterElement, there were no problems. The final test was for the entire filter functioning on it’s own. The Filter was fed a query, after which the Parser was invoked, the filter looked up the right FilterElements and called the appropriate methods. The result from the filter elements was then processed by the Filter class to apply the AND, OR and NOT operations. Here we again encountered some small bugs, but no problems of big significance.

6.3 Core

The Core module didn’t need any testing. The only methods used by NewProfiler are two get-methods and a method to invoke the flush function in the Cache. These are trivial and if we would manage to make a fault, it would be detected as soon as NewProfiler starts. The other methods are used for starting the selftest and the cachetest.

6.4 Contact

The Contact module was fairly easy to test. Not all of it’s methods are implemented yet, because the set-, create- and removeContact methods have to be custom built for every web shop. Which leaves the getContact and getContacts methods to test. Both tests were completed without much trouble. One fault that showed up was in the getContacts method. A NullPointerException would be thrown if no filter was contained in the received SOAP message. This turned out to be an implementation fault that was in all methods invoked by SOAP Messages using the filter. Furthermore, we also tested the getContacts method with the filter. This unveiled a problem with included contacts. These contacts would be returned with only the id field. Again this was an error that was easily fixed. Performance of this module was a bit disappointing, since it accesses an external database instead of the cache to get the information needed. It is highly probable that performance will not be as bad as we have seen when NewProfiler is deployed, since in this case, the database was running on a machine with low processor capacity and a slow internet connection.

Then there’s still a part of the Contact module that has not been tested. The Conversation module was dropped from the design and it’s methods are now in the Contact module. We skipped testing these methods for now, to save some time, because of the tight schedule we were on during testing. The functionality that these method’s provide are not essential for NewProfiler’s inner workings. And the methods in the cache that handle conversation were already tested. This is because the cache test relied on these methods. So we know for sure that those are working.

6.5 SiteMap

The SiteMap module was the first module tested that would actually use the cache for requesting and storing information. The test invokes all SOAP methods of the Sitemap module. Modifying the Resources and Folder worked as it should. We also launched some dirty commands at the module. For example, we tried to move a folder to a subfolder of
this particular folder. This was detected, and as the system should, it refused to make the move. If it would have continued an infinite loop would occur when searching the virtual filesystem. A problem we encountered was that the command to search the local filesystem for changed resources, is operating system dependent. Since a *NIX system doesn’t know a mount point starting with c: it will throw an exception when seeing this. This is not so much a bug of NewProfiler, but just a little shortcoming of java. Still, it won’t be a problem when deploying NewProfiler. The location of the NewProfiler directory will be given in a configuration file.

6.6 ContactGroup

The testing of the ContactGroup module turned out to be a fairly simple procedure. There were a couple of small errors which were easily fixed, and output was exactly as expected.

6.7 Mailing

The Mailing module test was the same as the ContactGroup test for a big part. The get-, set-, create- en removeMail methods were working immediately. Though the getMails method threw a NullPointerException when filter settings weren’t provided. Because we encountered this problem earlier in the Contact module, this was an easy fix. The biggest problem with testing the Mailing module was the sendMail method. First of all we needed a dummy SMTP server, so we wouldn’t be blocked because of sending spam. After a short search we found a java program that could emulate a SMTP server for us.[6] After invoking the sendMail method, at first everything seemed to be alright. The Mail messages were coming in on the fake SMTP server. After the ‘sending’ was done we inspected the sent mails and encountered a problem. The placeholder variables were still in place instead of the data they should have been replaced with. It took a while for us to locate this problem, but after a while we discovered that we used faulty regular expressions for replacing the placeholders. After this, it was quite easy to solve the problem.

6.8 Report

The test of the Report module consists of three parts:

1. The Report module is tested using a special prepared database. This test ensures that the Report module will produce the desired output and that it can handle a large database. Also reporting with a filter restriction and a time window is tested.

2. The Logger is tested for logging requests correctly including logging requests on unknown resources and tracking referrers correctly.

3. The Logger is tested for tracking sessions and contact logins correctly.

Each of the three tests requires a different test setup.
1. The NewProfiler database was filled with requests, resources and queries copied from SimpleReporter. This small brother of NewProfiler had been working for a few months on logging requests and the data could be converted to the format used by NewProfiler. We used this data to test NewProfiler and to compare it with SimpleReporter.

2. We created three web pages that only contain a link to each of the pages. The links cause the browser to go to another page and use the original page as a referral. Also the original page puts a query in the request so the query can be logged.

3. We created a login servlet. This servlet creates a small web page where the user can log in and out. This allows the logger to track contacts and sessions.

Each of the tests were run with both small and large time windows. The small time windows reduce the amount of output generated by the program. We compared that output against the database contents and against our activity during the logged period. The large time windows produce a lot of output and it is impossible to check the output by hand. We used the large time window to benchmark the system performance. The results from the tests are as follows:

1. The first tests pointed out that some of the auto-generated queries were broken. This problem was fixed quickly by manually testing the queries with an SQL administration tool.

The small-scale tests pointed out that the report functions work correctly. Also the logger had successfully logged the test invocations so it automatically keeps track of when we tested the application.

The large-scale tests pointed out that some of the queries were too slow. Some poorly-constructed queries caused a full-table scan on the session table (approx. 8,000 entries) for every request (approx. 12,000 entries). The SQL server we used did neither have enough memory nor processing power to answer the query within a reasonable amount of time (approx. 2.6 seconds). We rewrote some of the queries to use joins instead of sub-queries. We also specified a much smaller time window so it would cover only one month instead of one year. To our surprise, the report did not return any results at all. In one of the months (the month where we created the database), the program could not execute the query. Just like we experienced with the large time-window. It turned out that all of the request time stamps had been reset to the moment that we converted them from the SimpleReporter database. After randomizing the time stamps, the program did work correctly. We also learned that the test database actually approached the worst-case scenario for the logger.

The weaknesses that we found in the current implementation can be summarized as follows:

- About 90% of the requests in the test database had a new query. This caused NewProfiler to create both a query entry and a request entry for the request,
resulting in much database overhead. A one-to-one mapping of queries and requests is actually the worst case for the logger. See chapter 9 on how to address this problem.

- Some queries cannot use the local cache. Looking up requests and contacts requires database communication and can be slow.

- Specifying a large time window can put a heavy strain on the database server. If the request takes longer than a few seconds, the user will probably disconnect (e.g. close his browser window). The query will keep running even if no one is listening anymore. Although the Desktop connection pool will provide new connections if the old ones are kept open beyond their expected life span, the active database connection is maintained and might require the intervention of an administrator to be terminated. Also see chapter 9 about a solution to this problem.

2. The web pages in the test suite had already been found by the SiteMap after scanning for new resources. The logger did successfully log requests on these pages during the tests. Reporting about these pages worked flawlessly.

3. The logger automatically discovered the servlet when it was requested for the first time. Because the servlet is a virtual resource, it was not found by the SiteMap. The logger added it to the SiteMap as a virtual resource. It also logged out requests on this servlet. The servlet ends and creates sessions as well as contact logins. The logger did successfully log the requests done by our test contact account. The report contained exactly the path of resources we requested after logging in.
7 Results

7.1 Overview of project result

7.1.1 Features implemented

This section is a list of the features that were implemented.

- **Filter**
  - Parser implemented, it prepares the search query for processing by the filter
  - Filter works entirely.
  - Filter implementation can be easily modified to support more search fields.
  - FilterElement. A couple of generic filter elements have been implemented.

- **SiteMap**
  - SiteMap can search for resources on the local filesystem and add them to the database.
  - Resources and folders can be managed completely.

- **Logger** implemented as documented.

- **Contacts**
  - Methods for displaying contacts have been implemented.
  - Integration with filter has been completed.

- **Report**
  - Reports can be generated. There are separate methods for contacts, referrals, resources and requests.
  - Filter integration has been done.

- **Mailing**
  - Mailings can be created, modified and removed.
  - Mailings can be sent to a saved targetgroup.
  - Mailings can contain placeholder values which are replaced by the actual values before sending.
  - Requesting the overview of Mailing can be done with filter settings, this displays the Mailings sent to the Contacts that are returned by the filter

- **ContactGroup**. TargetGroups can be managed completely.
• Conversation implementation has been moved to Contact module. Conversations can be managed completely.

• First scalability tests indicated that speed is good. The tests used real data from another NewNomads project.

### 7.1.2 Features not implemented

This section is a list of features not (yet) implemented:

• Filter. Not all FilterElements described in the analysis document are implemented. The elements not yet implemented are not available on all web shops. Because of the generic filter implementation, these can be added later very easily.

• Contacts. In the analysis document we mentioned that NewProfiler can use its own database on top of an external database to provide extra contact fields. This is not yet implemented and probably never will be, since all external databases NewProfiler will work with, are designed by NewNomads.

• Report. The export functions are missing. These were skipped because of time limits and they were not a major feature.

• Order module has not been implemented.

• Help is not implemented yet, since there is no client.

### 7.2 Timeline

We started the project during the last quarter of the academic year 2006/2007. We started a few weeks late so we planned some additional weeks during the summer holiday. Compared to the original planning we managed to keep up until mid implementation phase. We also started testing some parts of the implementation after two weeks, as planned. As described in the implementation chapter (Chapter 5), we implemented the Cache and Filter first. The first tests were successful and we continued with the implementation of the modules.

We also started implementing the JavaScript client as planned. During a meeting with NewNomads, we decided that the JavaScript client would be discarded after the project and replaced with a Flash client. Because all work that would be put into the JavaScript client would be for nothing eventually, we decided to stop working on that program. Instead we would continue development on the server while NewNomads started building a Flash shell where we could create test scripts. By dropping the JavaScript client, we lost a few days of work.

Mid-summer the project was running late and the end was not anywhere near. The project was forced to the background due to other activities and a few problems. Completing the server side of the system alone took already all of the time we had reserved for this project and we still had to implement the entire client. This foresight was not very
motivating. Furthermore, a few hardware problems, unexpected side-activities and a lot of RSI caused us to stop working on the project for a while.

After the new academic year started we planned less time to spend at the project, since other courses we wanted to attend were starting. We worked at separate locations during that period, and as a consequence of that we had a lower productivity than normal. During the first quarter of the new academic year, implementation progressed gradually. We had meetings with our supervisor, but later on they were less frequent, because of lack of progress. During the second quarter we didn’t have one single meeting with our supervisor. We were busy waiting for the client and in the meantime following other courses. When not working on the courses that year, we were busy doing other work. Some new features for the Desktop were created because those were also needed for NewNomads. Some of these features are now also used by NewProfiler. We re-designed the Report module because the initial design did not provide a good basis for the implementation. The Home report was dropped from the design because it was poorly designed, might not work with the Flash client at all and because we found a better and more generic way to put it all in one Report module.

In February we sent a mail to our supervisor with apologies for the lack of contact. We scheduled a meeting to talk about finishing this project as soon as possible. Ir. B.R. Sodoyer was also attending this meeting and announced that we should decide on a deadline for finishing this project. Also we should try to finish it without being dependent of progress of the Flash client. We made a new planning for testing and finishing the required documentation. We got a SOAP tester program from NewNomads and used it to invoke the tests on the server side. Testing took us about 4 weeks, which was one week longer than we had scheduled. During testing we started working on the final report for the bachelor project. After a consultation between us and our supervisor, we moved some deadlines one week later.

7.3 Reflection

We were able to put our skills to good practice during this project. Many of the things we learned during previous years were put to use and we learned many new things.

7.3.1 Planning

During the initial planning, we did not have a good overview of what was to come. We simply planned a few weeks for each of the phases and a little overlap between those phases. The planning was adjusted after the analysis. It turned out that we had a poor overview of the amount of time this project was going to need. The project was bigger than expected and the planning was set up poorly due to lack of experience. The poor planning did not work very motivating.
7.3.2 Analysis

During the analysis phase, we met frequently with our contractor. This phase went according to plan and the analysis was set up properly. We were able to put our experience from other courses to good practice and the result was a very useful requirements document. We used this a lot during the rest of our project.

7.3.3 Design

During the start of this phase, we decided that we would not design all features in detail. Writing out all details is cumbersome, a lot of work and it takes much of the fun out of the implementation. We decided that the document would only contain the layout and purpose of each component of the system, and a detailed specification of any communication interfaces between the components. This setup proved to be a good balance between the amount of work in the design and implementation phase. The design document was very useful and will remain useful even after the entire system has been implemented. During the design, we had a lot of discussion about how to design many of the features. There was a lot of discussion about how the server needed to handle the data and how the client would display that data. It would have been much easier if we had split the server and the client software into two smaller projects. These systems are independent from each other and they only need a good specification of the communication interface in order to work together. The next time we would create such a system, we will create a design for the server and specify it’s interface to the outside world, and we will create a completely independent client design, based on the server interface. This solution splits the general problem into two smaller problems and provides better restrictions to which both systems must be committed. This probably saves a lot of time.

7.3.4 Implementation

As described in chapter 5, we started with the core components Cache and Filter. The only problem was that developing the Cache took more time than the Filter. While Ruben was still working on the cache, Martijn started working on some of the modules. The problem was that the modules cannot be built without the cache. Our solution was to create stub methods for parts of the cache that were not yet implemented. This was the only case where we had any problems with our approach of the implementation. In general, the implementation phase went very good. Near the end of the implementation, the development slowed down due to poor management of the tasks. In general, we are satisfied with our approach of this phase. It proved to be very effective to work this way. This solution might also scale with the size of the project and the size of the team. In a bigger project with more team members, we probably would choose to work with small groups of people in parallel.
7.3.5 Testing

The project has been tested in various phases. We already had a good framework to build NewProfiler on so many tests were already covered by the framework. Most input & output services for SOAP management, database connections and stability of modules are already tested during the development of the framework. This saved us a lot of time.

Our approach was similar to that of the framework: First build a solid base for the application, then extend it. We developed the Cache and Filter first. These components are the base for most modules. We tested these components thoroughly. The tests covered all of the features of these components such as asynchronous calls on the cache, complicated queries for the filter etc.

Next, we tested the modules. We tested these modules using ‘big-bang’ testing. This test method can be very effective but it can also become very complicated if a problem occurs. The tests went quickly and smoothly (of course, there were problems) and we are satisfied with the outcome. The reasons why this test method was very effective are the following:

- The Desktop framework already covered many tests and problematic situations for us.
- The Cache and Filter had been tested thoroughly so all complicated components were working properly.
- The error catching and reporting was implemented carefully. Problems could be located and resolved quickly.
- Clear specification of input and output in the design required us to structure the system. The code is easy to maintain.

7.3.6 Communication

Communication proved to be a bit problematic for us. We did contact our contractor frequently but sometimes forgot or simply did not have time to contact our supervisor. In the new academic year, we lost contact with our supervisor entirely.

The design document contains a lot of information that is usable for later reference. Finding the information you need in this document might take some time and this document might need to be restructured if it is updated with new information about new versions of the software. The quality of our documents has improved over time and newer documentation will continue to improve. In such documents, good summaries, reference pages and definitions need to be set up properly. So although the documentation is very useful already, a lot of improvements are still possible.

The next time, we should put communication on the same level as the implementation. More structured and systematic.
7.3.7 Project management

In general, we miscalculated the amount of time needed for the project. In order to complete both the client and the server application in good condition, about twice the planned time would be needed. Both the client and the server are a separate application which need to be built from scratch and they do need to work together. If we had focussed on the server only, we would have completed it within the planned amount of time.

The next time, we should split the client and the server system into a separate project and only specify the communication interface. This reduces the size of problems and forces us to design one component at a time. The importance of good equipment, a good place to work and the importance of working long shifts also should not be underestimated.
8 Conclusion

The current implementation of NewProfiler server is an excellent system for logging and reporting user activity. It can send e-mail messages to groups of contacts and measure the response, report about activity on the website and offers a lot of other features. Although not all of the desired features are implemented and there are a lot of additional features possible, this version is already a stable and scalable implementation of the most important tasks. Due to extensive documentation and a high quality of code, we expect that this application will be of much use to NewNomads and that it can be developed further if desired.

We learned a lot from this project. We put our knowledge to good use in building this system and we have gained some more experience in constructing a system of this size. Even more important is that we learned a lot from the many mistakes in planning and project management. The project took a lot longer than desired and expected. Our project management will be much better next time.
9 Recommendations

This section contains many items that we would recommend NewNomads to review after this project has been completed. It contains solutions to various small problems, suggestions for improvements on both NewProfiler and the Desktop framework and tasks that have not been completed during the project.

9.1 Incomplete tasks

- Needless to mention, a client application needs to be built.

- There is no Order module. This module must be implemented for the first customer who requires orders to be monitored by NewProfiler. This should be a generic implementation if possible.

- The Cache has failed its tests of the asynchronous mode. It is currently running in a slower mode which is safer. We recommend that the tests inside the Cache are run again, the problem needs to be identified and fixed. Then the cache can be used in a much faster mode.

- According to the design document, the SOAP methods can handle multiple tasks in a single request. In the current implementation, only the first task is executed. The method must be invoked for every task separately. For example, multiple resources can be specified in a ‘removeResource’ request. The method should be applied to each parameter. In the current implementation, the method is only applied to the first parameter.

9.2 Problems and solutions

- Generating reports requires a lot of memory. We recommend that at least 512 MB of RAM is made available for NewProfiler and the SQL server. Depending on the size of the report and on the amount of contacts, more memory might be required. It is also a good practice to limit the time window when requesting a report.

- If the program is shut down while a batch of e-mail messages is being sent, the batch is interrupted. Although the program keeps records of who has received the message, a resume option should be implemented to fix this problem.

- If the queries in the Report module get too slow once the database grows, the following measures can be taken to improve performance:
  
  - Increase the amount of memory allocated to the database server.
  - Rewrite the queries to perform their tasks in a different way. SQL might be able to optimize the queries if indices, joined tables or sub queries are used in a different way.
– Try to move some of the report processing to either an SQL script or to multiple small queries. This might solve the problem if a single big query becomes too heavy for the server.

• Logging the requests introduces a lot of overhead if each of the requests contains a different query. This causes NewProfiler to store the query and the request both as a new entry in the database. If queries and requests often get this one-to-one relation, we recommend storing them in a different way. For example, NewProfiler could offer the choice on how to log the requests on a resource: queries and requests as a many-to-one or a one-to-one relation.

• If processing the data becomes too difficult, it might help to store parameters using an additional level of abstraction. The queries on a resource often consist of multiple parameters in the form ‘param1=value1&param2=value2&...’. It might help to store these parameters in a key-value table.

9.3 Additions to NewProfiler and the Desktop

• The current logger can only log traffic on the servlet context it is running on. If resources are placed outside the context, the logger might never see traffic on those resources. The program does not support multiple loggers or logging at another level. This might be necessary depending on the server structure.

• The Contact and Order module can be changed to a more generic version that can be adapted from the configuration file. The current implementation can only be adapted by overwriting the module with a customer-specific version. A more generic version of these modules can be built to improve the re-use of code.

• A different logging mode can be implemented to reduce database size. For resources that are less interesting to report about, only the amount of hits should be recorded, not the entire request. Or alternatively, only the request time stamp needs to be logged, no additional information. This information could still be used to reconstruct the click path of the user or to report about the popularity of the resource, but it consumes less space in the database. The full logging mode should only be used on index pages or resources that are used during transitions between parts of the website.

• The modules can be split into SOAP-handling methods and internal methods. Only the SOAP-handling methods need to be put inside the module, other methods can be moved to separate classes. This will increase code reusability.

• Additional methods can be added to the SiteMap to enable the client to request an overview or details of queries.

• Conversations can be separated from the Contact module.
• Similar to the ‘searchFilesystem’ method in the SiteMap, a new method can be created to search for and remove missing resources.

• We recommend that after the first version for a specific customer has been built (including client program), the system is tested again. Tests should focus on boundary values and invalid characters that can be put into the system. Because our simulation of SOAP calls has not covered tests on parsing SOAP messages, it is possible that problems occur during the conversion.

• The Desktop knows about the required input and output of the services. It should use this information to verify correct behavior of the system.

• The connection pool should automatically close any connections that have been kept open for more than twice the desired lifespan of a connection. This will prevent queries from running for too long on the database server.

• The Filter should be extended to support filtering of resources or folders. The current implementation does only support filtering on fields of the contact table. Although creating output restrictions based on other criteria is possible, it requires multiple requests and sorting of report output.

• When removing a folder, the contents of that folder are moved to the parent folder. We recommend that an alternative remove folder option is implemented that removes a folder and the contents.
References

A  Installation
B  Project description
C  Analysis document
D  Design document
Appendix A:

Installation
Appendix B:

Project description
Appendix C:

Analysis document
Appendix D:

Design document