Collecting Actual Use Events from Browser-based User Interface - Model and Implementation

Pekka Partanen

May 30th, 2003

Master’s Thesis

Helsinki University of Technology
Department of Computer Science and Engineering
Usability School

HELSINKI UNIVERSITY OF TECHNOLOGY | **ABSTRACT OF MASTER'S THESIS**

Author: Pekka Partanen

Title: **Collecting Actual Use Events from Browser-based User Interface - Model and Implementation**

Date: May 30th, 2003 | **Number of Pages:** 107

Department: Department of Computer Science and Engineering

Professorship: Tik-121 User-centered Product Development

Supervisor: Prof. Marko Nieminen | **Instructor:** M.Sc. Seppo Tolonen

In this thesis actual use logging from a graphical user interface is studied. For this purpose, a model and a software implementation for collecting the actual use logging events from a generic browser-based user interface are developed. The model utilises the logging features of the web server so that the studied software need not be changed.

By collecting the user activatable elements from the user interface description, it is possible to form a hierarchical map of the interface. The intentional user actions are filtered from the collected and abstracted weblog by combining it into the hierarchical map. This thesis presents three analyses, which process the single user interface events and transform them to semantically rich information.

The user interface events provide an interesting insight into the user in the application context, and the aim of this thesis is to demonstrate how these events may be collected and analysed. The collection model presented in this thesis may make it easier to collect the user interface events so that they can contribute to the user interface design process. Within this thesis, the software implementation is published under the GNU GPL license.

Keywords:

actual use logging, GUI, usability

TEKNILLINEN KORKEAKOULU **DIPLOMITYÖN TIIVISTELMÄ**

Tekijä: Pekka Partanen

Otsikko: **Käyttäjän toimintojen kerääminen selainpohjaisesta käyttöliittymästä - malli ja toteutus**

Päivämäärä: 30. toukokuuta 2003 | **Sivumäärä:** 107

Osasto: Tietotekniikan osasto

Professuuri: Tik-121 Käyttäjäkeskeinen tuotekehitys

Valvoja: Prof. Marko Nieminen | **Ohjaaja:** FM Seppo Tolonen

Tässä diplomityössä tarkastellaan graafisesta käyttöliittymästä tehtävää käyttäjän toimenpiteiden keräämistä ja analysointia. Tätä varten kehitetään malli ja siihen perustuva ohjelmistototeutus, joka mahdollistaa toimenpiteiden keräämisen ja analysoinnin yleisestä selainpohjaisesta käyttöliittymästä. Malli hyödyntää verkkopalvelimen lokitoimintoja, jolloin keräystoiminnot eivät vaadi tutkittavan ohjelmiston muuttamista.

Keräämällä sovelluksen käyttöliittymäkuvauksesta käyttäjän aktivoitavat elementit voidaan käyttöliittymästä muodostaa hierarkkinen kartta, jossa sivut liittyvät linkkien avulla toisiinsa. Yhdistämällä kartta verkkopalvelimen keräämään ja abstraktoituun pyyntölokiin saadaan käyttäjän tekemät käyttöliittymätoiminnot suodatettua esille muista lokimerkinnöistä. Tässä työssä esitetään kolme analyysiä, joilla yksittäiset käyttöliittymätoiminnot saadaan muutettua semanttisesti rikkaaksi informaatioksi.

Käyttöliittymätoiminnot tarjoavat mielenkiintoisen näkymän käyttäjään sovelluksen kontekstissa. Tämän diplomityön tarkoitus on osoittaa, kuinka näitä toimintoja voidaan kerätä ja analysoida ja saada näin uutta tietoa sovelluksen käytöstä. Tässä työssä esitetty keräysmalli voi osaltaan helpottaa toimintojen keräämistä ja näiden aktiivista mukaanottoa osaksi käyttöliittymäsuunnitteluprosessia. Toteutuksen lähdekoodi julkaistaan GNU GPL lisenssillä.

Avainsanat:

käyttöliittymätoimintojen kerääminen, GUI, käytettävyys

Acknowledgements

This thesis was written in Comptel Corporation and it concludes my studies for a Master's Degree at Helsinki University of Technology. Since the beginning of this research, many people have made direct or indirect contributions to it, but only those with direct influence are mentioned here.

I would like to thank my thesis supervisor, Professor Marko Nieminen, for his optimism towards the possibilities of the actual use logging approach and my work. I would like to thank my thesis instructor, Seppo Tolonen, M.Sc., for his guidance and comments. I would also like to thank my colleague Roni Hursti, B.Eng., for his contagious enthusiasm towards free software and software development. I am also thankful to my helpful office mates, past and present.

Finally, I am very grateful to my parents Asta and Keijo in so many ways, as well as to my sister Eeva for her support. Without your reminding me about this thesis, I would have lost between my other interests.

Helsinki, May 30th, 2003.
Pekka Partanen

Contents

List of Figures

List of Tables

Glossary

Accelerator Key The accelerator key is a shortcut to some function or feature in the user interface. Usually it is a keyboard combination that accelerates the use of a particular function.

API Application Programming Interface. A set of services that a system is providing for applications.

AUL Actual Use Logging. Refers to collecting user actions through the user interface.

BSS Business Support System. A system deployed by a service provider to support business operations.

CSS Cascading Style Sheet. A technique used in WWW, which allows authors and users to attach style, e.g. fonts, colors and spacing, to HTML documents.

DB Database.

GSM Groupe Spécial Mobile. An open digital technology that uses time division multiple access transmission methods. Used in mobile networks.

GUI Graphical User Interface. Refers to techniques of using graphics, keyboard, and mouse to provide an easy-to-use, user interface to an application.

HTML Hypertext Markup Language. A publishing language for global distribution.

HTTP Hypertext Transfer Protocol. Application-level protocol for hypermedia information systems.

IP Internet Protocol. A protocol by which data is sent from one computer to another on the Internet. Each computer has at least one IP address that uniquely identifies it from the other computers on the Internet.

IT Information Technology

Javadoc A software by Sun Microsystems for generating application programming interface documentation in HTML format from a specific javadoc comments in source code.

MDS/SAS Mediation Device Solutions - Subscriber Administration System. A software product of Comptel Corporation for managing the data on subscribers and their services in a communications network.

Mediation The process of collecting, modifying, correlating and rating event records and delivering them to the BSS systems.

Provisioning Setting up a telecommunications service for a particular subscriber in a switch.

UI The combination of menus, screens, keyboard commands, mouse clicks, and command language that defines how a user interacts with a software application.

URI Universal Resource Identifier. Specifies an abstract or physical resource (e.g. a web page). Includes URL or URN.

URL Universal Resource Locator. A subset of URI that identifies a resource through its primary access mechanism (e.g. its network location) rather than by name.

URN Universal Resource Name. A subset of URI. Must remain globally unique and persistent even when the resource becomes unavailable.

WWW World Wide Web. A network of information resources.

Chapter 1

Introduction

1.1 Background

The development of software should be based on real information about users' needs and on their key interests in the current software products. For example, information about the most frequently used application features can suggest which features to optimise and how to focus product development in the near future. (Hilbert & Redmiles 1998)

User interface (UI) events are generated as products of the user operations in the computer systems. They provide a good insight into the user's behaviour in a real environment, when the user is doing his actual work (Nielsen 1993). User interface events can be collected automatically and they can be used in different statistical analyses. Typically, the collected information contains data about the UI events so that the frequency with which each user has used different features in the program is possible to calculate. Furthermore, the frequency with which various special events (such as online help accesses) have occurred might also be included in the collected information. That kind of information can be used to make analyses e.g. about the software features that most and least frequently need online-help. Usability of software can be improved by optimising the most frequently used tasks and prioritising the usability problems of different tasks by basing prioritisation on task frequency analysis. A good advance in usability is achieved by improving the most frequently used features.

Actual use logging (AUL) approach has been described as a fruitful source for usability evaluation at least since 1982 (Buxton, Lamb, Sherman & Smith 1983). However, little empirical work has been performed to evaluate different actual use logging approaches' strengths and limitations. (Hilbert & Redmiles 2000)

The AUL approach does not remove the need for qualitative usability research methods like user interviews. AUL provides quantitative approach to usability and gives street credible data for the user interface developers. But before AUL can be applied, it needs an already existing software system. So, the best advance from AUL is gained by integrating it to the qualitative usability methods without thinking it as a competing "new school" approach.

This thesis started as a usability research project to Comptel Corporation. The company wanted to get profound information of the usage of one of their software products; there was no information about how the users use the software in real life. The company did not set the method for solving the usage, and the decision of the methodology was left for the thesis writer. After studying usability literature and the company's software product, AUL approach was chosen to be applied. Even though the thesis is centred on developing a model and implementation of AUL software tool, the basis of the original project was built on the company's real life problem.

Comptel Corporation is a Finnish company, established in 1986. The company provides mediation and provisioning software for telephone operators and service providers. Comptel's products offer connections from business support system (BSS) to network elements. One of the main products is Mediation Device Solutions - Subscriber Administration System (hereafter referred to as MDS/SAS).

1.2 Goals

The goal of the thesis is to study different AUL approaches and provide a model and an implementation for a generic AUL collector and analyser tool for the browser-based user interface. The analyses are derived to give insight about how to focus software development efforts in the near future. For example, information about the most and least used user interface parts gives valuable knowledge of the

user behaviour in an application. The information is based on real use situations where the user's actions in the user interface are analysed.

This thesis concentrates on developing a working model for collecting and analysing AUL information for the needs of graphical user interface designers who want to have "hands-on" information of their systems. Effort is made to provide a highly configurable and extendable implementation, which can be used as a general tool in the user interface development projects. The application environment, MDS/SAS 5.0, acts as a case study in where the idea of the model and the actual implementation are tested.

According to the idea of AUL, this thesis does not give usability improvement suggestions for MDS/SAS 5.0. They would require qualitative research of the use cases and the software, which are not in the scope of the thesis. The user interface designers have knowledge about those issues, and they know how to integrate the results of the analyses to the software.

Chapter 2

Literature Review of Actual Use Logging

This chapter provides theoretical background for the following chapters. At first, a high-level separation of different methods is described. Then follows sections for the contextuality and the abstraction levels and, finally, an introduction to different interpretation techniques.

2.1 Actual Use Logging Approaches

In principle, there are two main approaches to do actual use logging, as described in Nielsen (1993). One way is to collect data with generic data collection software e.g. CHIME (Badre 1980). These kinds of software log fairly low-level user actions like button presses and mouse traces without any semantic information about what was being clicked or selected (Guzdial, Santos, Badre, Hudson & Gray 1994). At the same time, the volume of collected data might be great depending on the data collection period. The approach is generic so that the data collection software can be used to study the use of any software without any modifications to the studied software.

The other way to perform the collection is to modify the software of interest. The advantage of this approach is that the semantics of user actions can be included in

the collected information. In general, all interesting information can be collected for further analysis. In contrast, this method still has some disadvantages. First, the method requires access to the source code, which might be impossible to achieve. The second disadvantage is that it is difficult to decide the level of detail the UI events should be collected at. If the level is too low, the volume of data gets easily too large but if the level is too high, the desired data might not be available at the time of the analysis.

2.2 Contextuality of User Interface Events

Problematic issues arise when attempting to interpret the significance of the UI events based only on the information that the events carry by themselves. The context of the event is important because substantial contextual cues are often spread across the interface. This problem is analogous to natural language conversations. To illustrate the problem more generally, an analogous problem of interpreting the significance of utterances in natural language conversation is described. The example is from the study by Hilbert & Redmiles (2000).

There was a conversation between persons A and B at a car show. The task is to identify A's favourite cars based on the utterances in his expressions.

Example 1:"The Lamborghini is one of my favourite cars".

In this case, all information that is needed to determine one of A's favourite cars is contained in a single statement.

Example 2: "The Lamborghini".

Here, access to prior context is needed before determining anything. It is most likely that A is responding to a question posed by B. The information carried in the question is critical in interpreting the response. The question might have been like "what is your least favourite car?" or "what car does your rich uncle drive?".

Example 3: "That is one of my favourite cars".

In this case, the needed information was available to the interlocutors at the time of the utterance. The term "that" points to the needed information and de-reference

to that information is required.

Example 4: ”That is another one”.

This case requires access to prior context and the ability to de-reference the item to which the term ”that” is pointing.

All examples above are analogous to interpretations of UI events. The following examples illustrate the situations:

Example 1: `PrintToolbarButtonPressed`

There is enough information within the event itself so that it indicates the action the user has performed accurately.

Example 2: `CancelButtonPressed`

This event does not describe what the cancelled action was. Analogously to the Example 2 above, this event indicates a response to some prior event, e.g. a prior SaveFileAs and cannot be interpreted by itself.

Example 3: `ErrorDialogActivated`

The interpretation of this event needs information that may have been available in prior events. More direct way to interpret it would be to query the mentioned dialog for its error message. Specifically, this is similar to de-referencing the term ”that”, if we think that the error dialog is pointing to the error message that does not appear in the event stream.

Example 4: `ErrorDialogActivated`

It also might be possible that the error message was due to an ”illegal command”. In this case, the needed information is not found by de-referencing the indexical but must be combined with the information available in prior events.

The fundamental observation in these examples is that sometimes a UI event does not carry enough information to interpret its significance accurately. The needed information might be available in other events or it was available at the time when the event occurred. This has to be remembered when designing an event collection software. It has to be capable of solving or skipping the events that it considers as

being ambiguous.

2.3 Abstraction Levels in Analysis

User interface events can be analysed at different abstraction levels depending on the usability issues that are in the research focus. One may wish to analyse low-level physical things like mouse movements while the other may focus on higher-level task solving steps taken by the user. An example of a high-level task is an online settlement paying.

Low-level events have meaning only as quantitative data. High-level events can be thought to be qualitative data, because usually they are derived from quantitative data by some deduction rules. These rules require human intuition about the user and the tasks that he/she is performing. However, high-level events' represention is quantitative even though they are qualitative in the respect of user interfaces. Sometimes, high-level events have simple representations in single low-level events as described in the following paragraphs. In those cases, high-level events differ from low-level events only semantically.

It is noticeable that the lower the abstraction level is, the more objective it is. If we have to interpret the user's needs and goals based on his user interface activities, we have to assume a great deal about the decisions that he made, because we cannot collect all prior information. In other words, we have to add our own intuition to the objective data by giving a meaning to the event stream. For example, the actual use logging system has collected the following event stream:

```
AddSubscriberToDB SubscriberName = "Jack Smith"
AddSubscriberToDB SubscriberName = "Bruce Smith"
DeleteSubscriberFromDB SubscriberName = "Jack Smith"
```

When we interpret the stream above, we have to assume something about the intentions of the user. It might be possible that the user made a mistake in writing the user's name as "Jack" instead of "Bruce" and noticed the mistake after it occurred and removed the name from the database. On the other hand, it might be possible that the user received a message that "Jack Smith" closed his account after

the user added “Bruce Smith” to the database and there was no error in the user’s actions. All in all, there is always a compromise between the number of collected and available data. Even if user interface designers considered the different use cases and the user actions that relate to them, the users might find new ways to use software that the designers did not even consider are possible (Nielsen 1993).

Six different abstraction levels are assumed (Hilbert & Redmiles 2000). They are described in Figure 2.1. In addition to the abstraction levels, the level of intuition is described in the figure as well.

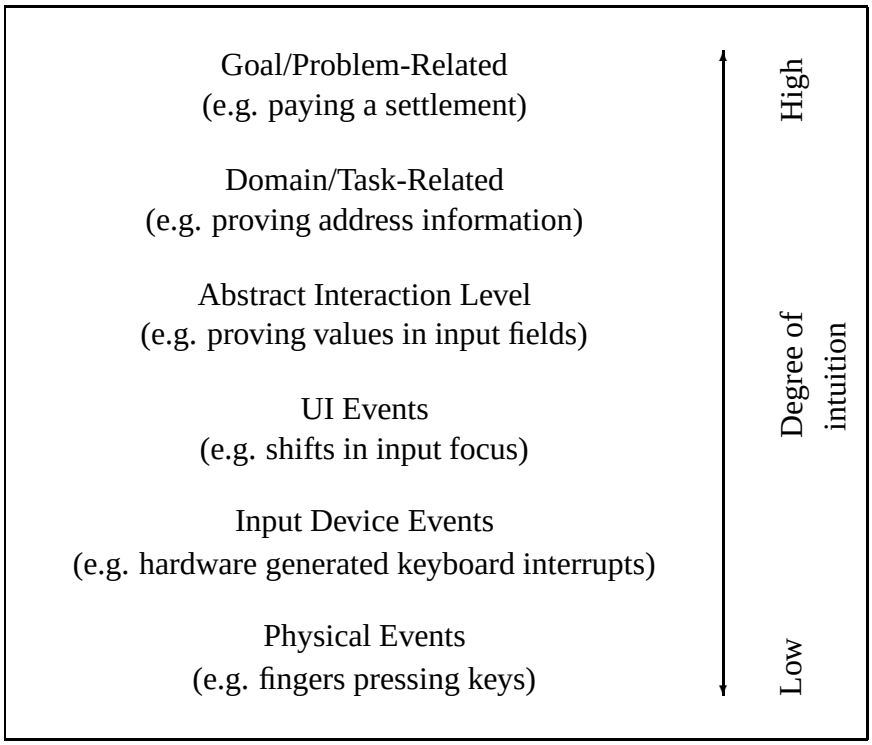

Figure 2.1: The abstraction levels of the user interface events. Events that are low in the figure need little interpretation and, therefore, little intuition from the interpreter. The higher the level, the more intuition the interpreter has to add to the pure information.

At the lowest level are *physical events* that could be e.g. key pressings or mouse movings. *Input device events* are generated by the hardware in response to physical events. These events could contain e.g. keyboard or mouse interrupts. *UI events* associate screen and other interface objects with input device events. Events at this level include e.g. focus shifts and radio-button selections.

Abstract interaction events are not directly generated by the UI system, but they may be deduced from UI events. The events themselves are typically indicated by idiomatic events. For example, when a user wants to edit a textbox, which is the last box in a group of text-boxes, he has to press tabulator repeatedly to change the focus to the right text-box. In terms of UI events, the focus shifted several times between the first and the last text box. In terms of abstract interaction events, the focus shifted directly from the top of the page to the right text-box.

Domain/task-related and *goal/problem-related events* are the highest level events. While the other levels consider the UI as the object of actions, these two highest levels take the user's tasks and goals as objects. Sometimes it is easy to infer these events from lower level events. For example, wizards in WinZip software guide users through a sequence of steps in a predefined task and the user's progress can be recognized in terms of simple UI events, such as button presses on the 'Next' -button. In some other cases, indicating the events from simple events may require more complicated composite event detection. For example, the goal of paying a settlement requires the task of providing information about the receiver of the settlement. This task can be recognized in terms of abstract interaction events (providing a value) occurring within each of the required fields in the receiver section of the form.

2.4 Interpreting Actual Use Logging Events

After the UI events have been collected, they have to be interpreted. The common nominator for the different interpretation techniques is that they allow interesting issues emerge from the "noise". Depending on the event collection period, there might be hundreds of thousands of events that need to be used as material for analysis. There has to be a method to automatically analyse the data.

The goal in the UI event analysis is to provide information from a great amount of material that is impossible to deduce from single observations. The following chapters describe the characteristics of different techniques that are synchronisation and searching, transformation, counts and summary statistics, sequence detection, sequence comparison, sequence characterisation, visualisation and integrated support (Hilbert & Redmiles 2000).

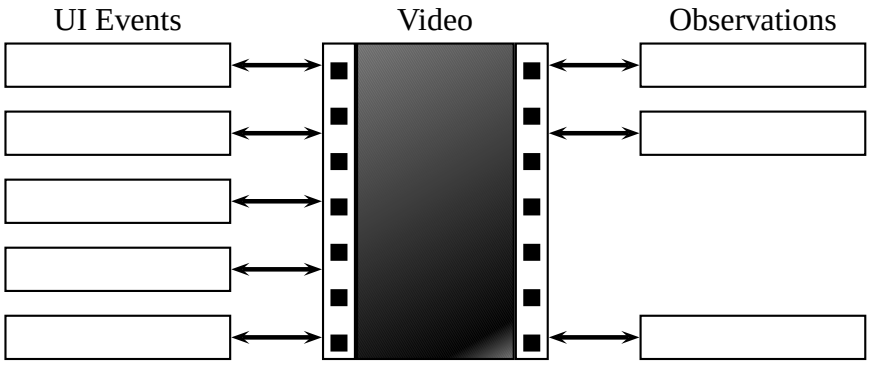

Figure 2.2: UI events are synchronised with video and observations. Investigator can do a search in one medium to locate the same event in other medium.

2.4.1 Synchronisation and Searching

These techniques are based on synchronisation and cross-indexing with UI events and other sources of data such as video and coded observations. The connections between elements are described in Figure 2.2. An interesting observation in one medium can be located and studied in another medium. Synchronisation and searching approach is the middle course between quantitative and qualitative usability evaluation. In addition to UI event collection, it uses observational evaluation to give more insight about the contextual information that is simply missing from the event stream.

The UI events are synchronised with video or coded observations so that if the investigator wants to review all segments of a video in which the user invokes a particular command, he does not have to do it manually. He can search through the user interface log for particular events of interest and use the timestamps, which are associated to the events, to jump to the right segment of the video recording.

This is the most simple technique but still powerful. There are many big IT companies (Apple, Microsoft, SunSoft) that have reported the use of tools that provide synch and search capabilities (Weiler 1993).

The strength of this approach is also its weakness. Observations need the presence of one or more observers or at least the presence of video equipment. The

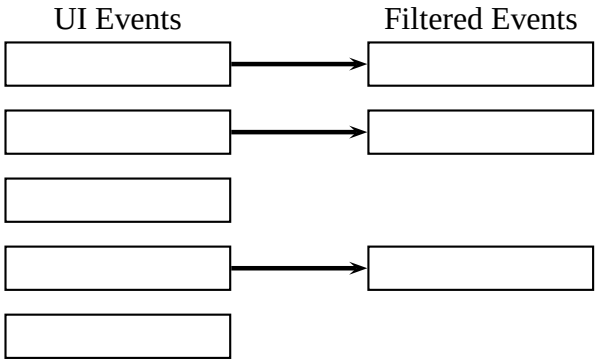

Figure 2.3: Filtering is one of the transformation techniques. It operates by selecting only the events that fulfill a described criteria.

situation becomes more laboratory like and it might affect the users performance. Furthermore, video analysis tends to produce a huge amount of data that takes lots of time to analyse. The ratio of the time spent in analysis versus the duration of the analysed sessions can easily be 10:1, as described in Sweeney, Maguire & Shackel (1993).

2.4.2 Transformation

This approach is actually a collection of different techniques. It combines filtering, abstraction and recoding to transform event streams for different purposes, such as comparison and characterisation. In characterisation, event streams can be categorised to various classes and compared to each other. Transformation is done in real time with the collection.

Filtering operates by describing a mask for wanted or unwanted UI events (Figure 2.3). One may filter out all events associated with mouse movements in order to focus on higher-level actions such as menu selections. Another way to apply filtering is to filter out information from user interface events that distinguish them from each other. For example, one may wish to filter out mouse coordinate values associated with button presses if they distinguish button presses within the same button.

Abstraction operates by combining new events based on lower-level events (Figure

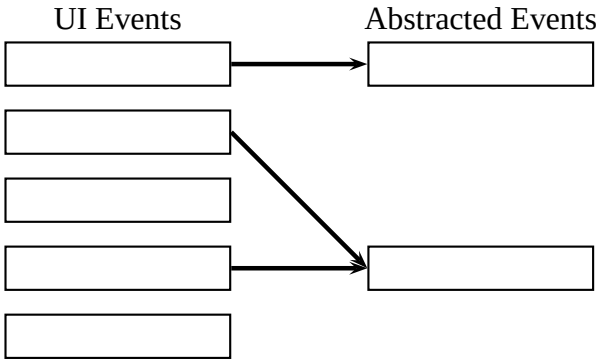

Figure 2.4: Abstraction is one of the transformation techniques. Abstraction operates by generating new events from existing events. Low-level events can be abstracted to higher-level events.

2.4). For example, events that indicate the opening of a pull-down menu, setting of the focus on a function and button press on the function might indicate the abstract event `'FunctionSelected'`.

Recoding operates by producing new event streams based on filtering and abstraction. Different low-level events may activate the same higher-level event and after abstraction they can be recoded to be the same events. For example, printing can be activated either by selecting 'File' menu and selecting 'Print' from it or by pressing the accelerator key (e.g. 'Ctrl-P'). After abstraction and recoding, both printing methods can be identified to be the same.

Transformation techniques limit the volume of collected data, which is usually a necessity for the collection. On the other hand, it is a limitation because of the risk of throwing away data that might have been useful in analysis.

2.4.3 Counts and Summary Statistics

After user interface events have been captured, there are a number of counts and summary statistics that can be applied to the data. These statistics contain feature use counts, error frequencies and use of the help system. It is possible to rely on general purpose analysis programs such as spreadsheets in analysing the collected

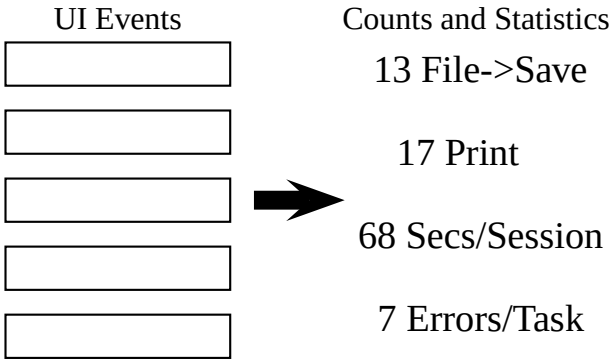

Figure 2.5: Counts and Summary Statistics are numeric values, which are generated from user interface events to characterise user behaviour.

data but more sophisticated systems are also available. The approach is described in Figure 2.5.

The MIKE user interface system provides built-in facitilies for calculating and reporting metrics (Olsen & Redmiles 1998). MIKE's operation is based on described user interface and application command abstractions. MIKE monitors UI events and associates them with interface components and application commands. Metrics that MIKE can report include the following:

- *Performance time* measures the time that was used in completing a certain task.
- *Mouse travel* indicates the distance between UI elements. The mouse travel should be low in relation to the number of elements.
- *Command frequency* describes the numbers of activated commands.
- *Command pair frequency* is related to commands that are used adjacently with some other command. Frequently used command pairs are better to be combined.
- *Cancel and undo* describes the commands that are frequently cancelled and undone.
- *Physical device swapping* indicates the switches between keyboard and mouse.

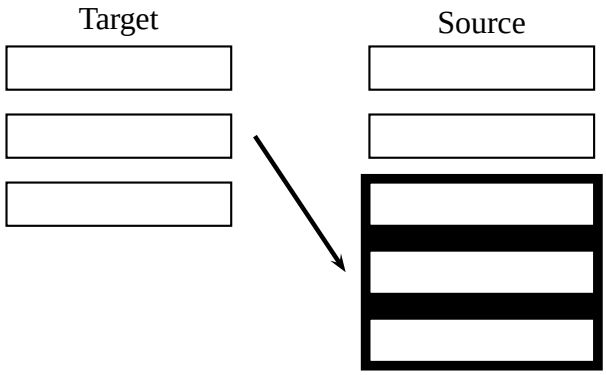

Figure 2.6: In sequence detection, concretely or abstractly defined target sequences are detected from source sequences.

The strength of this approach is that it provides a quantitative insight to user interface events. This reduces the possibility of misinterpretations. The limitation is that the approach does not use information that might be included in the context of a single event such as prior events.

2.4.4 Sequence Detection

This approach is used for detecting subsequences (target) from user interface event sequences (source). Subsequences can be abstractly or concretely defined. Abstractly defined sequences are supplied by the developers of the techniques (e.g. Fisher's cycles (Fisher 1991) and lag sequential analysis (Faraone & Dorfman 1987)), whereas concretely defined sequences are supplied by the investigators using the techniques (e.g. TOP/G (Hoppe 1988)). Figure 2.6 describes the approach.

Fisher's cycles are all subsequences that have common beginning and ending events defined by the investigator. The subsequences are identified automatically from the source sequence and their frequencies are reported. For example, the investigator might want to study what takes place when the user has activated the Print window and before he actually prints the page. In this case, the investigator defines the activation of the printing feature as a beginning event and the pressing of the OK button as an ending event. Fisher's cycles are all different event combinations that

take place in a real use situation between the defined events.

TOP/G is a command line parser that tries to infer higher level tasks that are being performed. Low-level commands are compared to task-action grammar, which is represented in a Prolog DB as rewrite and production rules. The rules are defined hiearchically in terms of elementary tasks. Keystroke level events are mapped into the tasks.

The strength of this approach is the ability to detect patterns of interest in events and not just perform analysis on isolated events. Combining sequence detection and transformation can produce “abstract” events from event streams. The limitation is that defining the patterns of interest might take time and limit the collected material so that new interesting patterns are not observed.

2.4.5 Sequence Comparison

These techniques try to find partial matches between the event stream and the abstractly or concretely defined target sequences. Some techniques attempt to detect the difference between a concrete target sequence produced e.g. by an expert user and a source sequence produced by some other user (e.g. ADAM (Finlay & Wolf 1995)). Points of deviation between the target and source sequences is still a common technique to find possible critical incidents (e.g. EMA (Balbo 1996)). The purpose in the sequence comparison approach is to compare the actual usage against some “ideal” or expected usage to identify potential usability problems. Figure 2.7 describes the approach.

The strength of this approach lies in its ability to analyse a specified part of a user interface. Only a small part of an interface can be modelled to a system, which causes the fact that only the events close to the model are “matched” and compared.

All these techniques are based on two assumptions that turn out to be limitations. The first assumption is that all interaction can be split to small segments and compared segment by segment. The second assumption is that interaction sequences are somehow similar between different users. If these assumptions are considered to be wrong, then the whole approach is questionable. In addition to the limitations

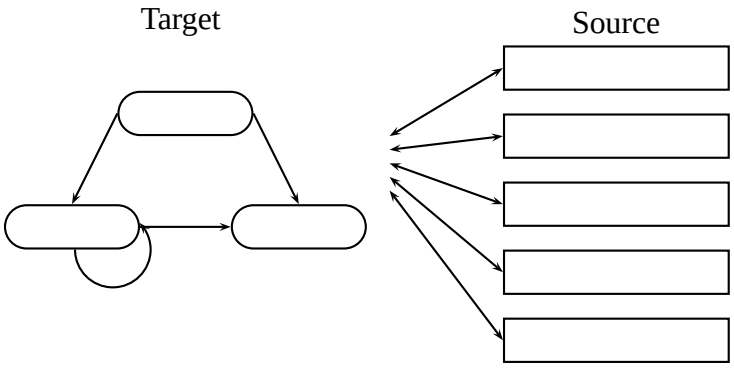

Figure 2.7: Sequence Comparison is comparison between the defined target sequences and source sequences. Sequences can be abstractly or concretely defined.

based on the assumptions, an expert interpreter is needed to determine whether the sequences are interestingly similar or different.

2.4.6 Sequence Characterisation

These techniques construct abstract models from user interface event sequences to summarise or characterise interesting features of the sequences. Some techniques construct a transition model with probabilities associated to transitions (e.g. Hawk (Guzdial 1996)), while others construct models that characterise the grammatical structure of the input event sequence. The approach is depicted in Figure 2.8.

Hawk system is based to Markov Chain analysis that can be used to produce process models with probabilities assigned to transitions between different states. A state of a system is considered to depend on the previous r-states in which case the system is called an *r*th-order Markovian system (Chen 1993). At first, the states of the system are defined. All events in the user interface event sequence are mapped to appropriate states. After that, they are abstracted and recoded to replace low-level events with abstract states. The transition probabilities are then calculated based on the UI event sequences. When the transition matrix is acquired, “one could use various existing techniques of Markov analysis to study the homogeneity of the distribution of probabilities, the symmetry of the transition matrix, the distribution of the process in a long-run, and the difference of the matrices under

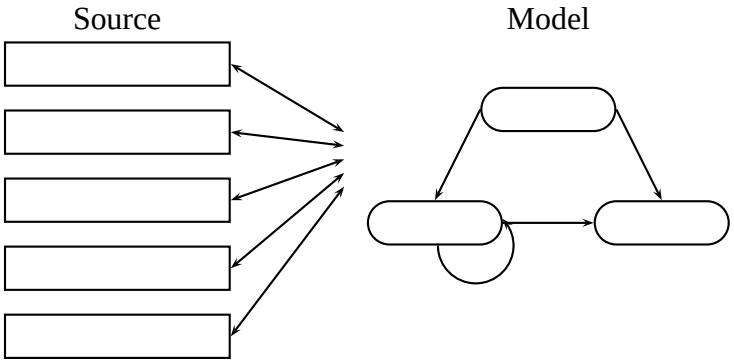

Figure 2.8: Sequence characterisation is for analysing the source events and generating abstract models for characterising the nature of the source events' sequential structure.

different experimental conditions" (Chen 1993).

The strength of sequence characterisation techniques is that they provide an insight into sequential structure of user interface event sequences. The limitation in the Markovian models is that it is based on a simple assumption that the current state is depending of the r previous state. However, Markovian models have been reported to provide useful information to investigators (Hilbert & Redmiles 2000).

2.4.7 Visualisation

These techniques allow human investigators to use their innate visual capabilities to order and to find relations in visually presented actual use log data. This approach is very informative when results are graphically linked with the interface.

The previously described approaches can be applied with visualisation so that the data is first e.g. transformed and abstracted and the results are presented graphically. For example, MacSHAPA (Sanderson, Scott, Johnston, Mainzer, Watanabe & James 1994) visualises the abstraction and transformation of low-level events to higher level abstract interaction events and task-related events in a spreadsheet. The visualisation is presented in Table 2.1.

Counts and summary statistics are commonly presented in the context of the actual

UI Events	*Abs.Interaction Events*	*Task-Related Events*
GotFocus(Name)		
Key(Name, 'T')	GotEdit(Name)	AddressSectionStarted()
Key(Name, 'e')		
Key(Name, 'x')		
LostFocus(Name)		
GotFocus(Street)		
Key(Street, '3')	LostEdit(Name) GotEdit(Street) ValueProvided(Name, "Tex")	NameProvided("Tex")
Key(Street, '1')		
Key(Street, '4')		

Table 2.1: Visualisation of the correspondence between user interface events at different abstraction levels

user interface. For example, a graph of mouse clicks can be presented on top of the interface so that the activity of different interface elements can be seen in relation to the other elements on the same page. Sequence characterisation can be combined with visualisation as well. For example, Hawk (Guzdial et al. 1994) presents a process model with nodes representing transition steps and arcs indicating the measured probability between process states.

2.4.8 Integrated Support

This approach is a combination of the previously described approaches. Different approaches are combined to make up an environment that enables composition of various transformation, analysis and visualisation capabilities. Only a single tool is needed to analyse very different types of information relating to actual use logging.

There are several different environments, which provide integrated support features like MacSHAPA (Sanderson et al. 1994), DRUM (Macleod & Rengger 1993) and Hawk (Guzdial et al. 1994). For example, MacSHAPA is a comprehensive environment designed to support several actual use data analysis. These features

include the following:

- data import and export
- video and coded observation synch and search
- user interface event filtering, abstraction and recoding
- built-in counts and summary statistics
- sequence detection, comparison and characterisation
- a number of visualisations and reports

The strength of integrated support approach is in its ability to handle the collection of usability information from the user interface and provide the analysis tools as well. This significantly reduces the data management and possible incompatibilities in data transfer between different programs. All integrated support environments have their own limitations. For example, MacSHAPA is not specifically designed for event analysis. Therefore, it requires a lot of human intervention and interpretation to extract useful information.

Chapter 3

Task Frequency tool

The practical part of this thesis is to develop a model and finally an implementation for both collecting and analysing the user actions in browser-based user interfaces. Attention is paid to design and implement a generic tool, which can be used in analysing different user interfaces.

Task Frequency Tool (TFT) is separated to two different parts, which are the collection and the analysis. The collection stores the user actions and provides them as input to the analysis. The model and the implementation are tested against a real software product in Chapter 4.

3.1 Technical Construction Elements and Boundaries for Collection

The problem in collecting the user actions from graphical UIs has been that the collection mechanisms have had to be implemented in the source code of applications in different log writing routines as in Emile (Guzdial 1996). Information about the user actions have not been available outside the applications for external collection. There simply has been no room for a generic user action analysis tool.

The following sections provide information of the technical construction elements and boundaries that are centric to the collector model. The model is presented in

Chapter 3.2.

3.1.1 Client-Server Model of Interaction

The primary pattern in the interaction between the cooperating applications is known as the *client-server* paradigm (e.g. Comer 1995). A general term *server* can be applied to any program that offers a service, which can be reached over a network. The server receives and accepts a request over the network, performs its service, and returns the result as a response to the requester. An example of the server is a web server. It receives requests for web pages and returns an appropriate page as a response. The web server could contain an application that processes the request's information and forms a page dynamically and returns it to the client.

A program becomes a *client* when it sends the request to the server and starts to wait for a response. In fact, the client-server model is an extension of interprocess communication on a single host. Thus, it is a natural and easy model to build programs to networked environments. An example of the client is a web browser. It sends requests of web pages to the server according to the user actions.

3.1.2 Hypertext Transfer Protocol

The user interfaces are transferred from the server to the client by using Hypertext Transfer Protocol (HTTP), which is an application-level protocol for hypermedia information systems (Net 1999). It has been in use in WWW since 1990. HTTP messages consist of requests from the client to the server and responses from the server to the client. An example of the server-client interaction model in the case of Internet browsing is depicted in Figure 3.1.

Selection of a hypertext link generates a 'GET' type request, which means "retrieve whatever information is identified by the identifier" (Net 1999). The resource identifier is a Uniform Resource Identifier (URI) that defines e.g. a web page. Therefore, a HTTP-request could be like the following:

```
GET http://www.foo.bar/pub/html/foo.html HTTP/1.1
```

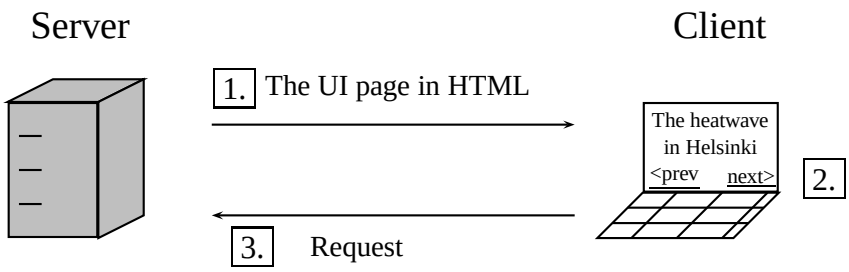

Figure 3.1: Server-Client interaction model in WWW-browsing. Server sends a UI page to client's web browser in HTML. The user clicks a link on the page, which generates a request for a new page to the server. The numbers indicate the order of steps.

The resource could be a web page that is a response to the user's selection in the UI. Also, it could be a servlet that reads parameters from the URI and returns a web page depending on the parameters. The parameter part of the URI is called a query-part and an example of the request containing a query is the following:

```
GET http://www.foo.bar/pub/servlet/user?id=ann&age=21 HTTP/1.1
```

The user does not have to know anything about the URIs that the application is generating from his actions in the UI. In HTML, the anchor element combines the user understandable text to URI. The meaning of the anchor tag is that it links the different documents, e.g. UI pages, together. The syntax of the anchor tag is following:

```
<A HREF="target">Anchor text</A>
```

The anchor text represents the text that is shown in the UI as a link and of which the user can choose by clicking it with a mouse. When the user clicks the link text, the Internet browser takes the *target* text and sends it in an HTTP request's query-part to the server. An example of the procedure is depicted in Figure 3.2.

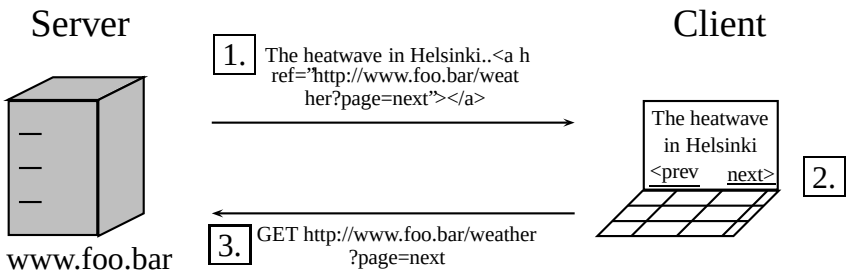

Figure 3.2: HTML in HTTP requests. Server sends a UI page to client's web browser in HTML. The user clicks a link on the page, which generates an HTTP request to the server. The numbers indicate the order of steps.

3.1.3 Web Server in Collection

When the user clicks a link in the user interface, the browser sends an HTTP request to the web server. The type of the requests is 'GET', which causes that the parameters are included to the URI.

Web servers have built-in functionalities for logging the requests that the clients are sending. They can be configured to write a log file entry for each HTTP request. Regarding the collection, the most important information in a log file is the query part of the URI.

Web servers support two standard-like log file formats, which are known as Common Log File and Extended Log File formats. They both provide the logging of the URI's query-parts and therefore only the Common Log File is presented here.

The Common Log File format is the following (W3C 1995):

```
remotehost rfc931 authuser [date] "request" status bytes
```

remotehost Remote hostname

rfc931 The remote logname of the user

authuser The username as which the user has authenticated himself

[date] Date and time of the request

"request" The request line exactly as it came from the client

status The HTTP status code returned to the client

bytes The content-length of the document transferred

A limitation in the web server log collection is that it does not provide features to log data that was sent with POST method. The limitation is that the content of POST information cannot be logged and therefore analysed later.

3.1.4 Mediator Requirements

In the case of mediator software (see Chapter 4), it is important that there are no interruptions while the system is in production use. The functionality of the software has to be such that the total throughput of BSS requests is as great as possible. If the requests are queuing in the mediator, it causes overload to the operator's system because the system waits an anknowledgement to each sent request.

The criticality of the system, as described above, implies two basic requirements for the user action collection. The first requirement is the fact that the total throughput of requests in the mediator has to be maximized. After all, the collection is only an extra feature and it should not affect the main task of the mediator in any way.

The second requirement is indirect. It says that there cannot be any extra modules in the mediator because a malfunctioning module could be very threatening for the stability of the teleoperator's network. At least, the extra module, which is the collector in this case, would have to be tested thoroughly in each system that it is going to be used.

After all, the risk of threatening the stability of the system and the amount of work in testing an extra module suggests that the collection should be done in some other way than with a module to the software core. Also, an extra module in the mediator would naturally decrease the throughput of the requests.

3.2 Model for Collection

As a result of getting familiar with the web-application environment and the Internet protocols, it was possible to draw a general model for collection. Even though the mediator requirements for collection were strict, they could also be overcome, but with a limitation to the collectable elements. It was decided to collect only the link information. The result is that the user navigation paths are raised into focus. At the same time, the model remains generic and does not need any extra features to the web server.

3.2.1 Filtering the Collected Usage Data

According to the HTTP protocol, a separate request is needed for each file that is wanted to be downloaded from the server to the client. As a result, the user's request to view a particular page generates requests for graphics and scripts in addition to an HTML file (Cooley, Mobasher & Srivastava 1999). All these requests generate a new entry to the web server log so that the intentional user action is concealed by the extra log entries.

Before the logged user actions can be used efficiently in computation, they have to be filtered from the data. This method was described in Section 2.4.2. Here, the filtering cannot be performed in real time because the collection is done on the web server. After the web server has collected a weblog for a desired period of time, the filtering can be started.

In order to do the filtering, the information of the user interface elements have to be known. Then, the insignificant weblog entries can be discarded and an analysis of the real user actions started.

3.2.2 Acquiring the User Interface Structure

Each HTML element type has its own description in HTML so that it can be recognized from the textual web page. The link structure of a web page can be acquired by parsing the anchor elements from the web page's HTML file. Both the anchor

and target texts have to be parsed from inside the anchor tag. Figure 3.3 shows in bold face the parts of the HTML page that have to be parsed from a web page snippet in Figure 3.4 for link analysis.

```
<HTML>
 <TABLE>
  <TR><TD>Name</TD><TD>Value</TD></TR>
  <TR><TD>Task ID</TD><TD>3</TD></TR>
  <TR><TD>NE ID</TD><TD>in2</TD></TR>
  <TR><TD>NE type</TD><TD>IN</TD></TR>
  <TR><TD>Task type</TD><TD>create</TD></TR>
  <TR><TD>NE completion date</TD><TD>2003-01-10</TD></TR>
  <TR><TD>NE submission date</TD><TD>2003-01-10</TD></TR>
  <TR><TD>Priority</TD><TD >5</TD></TR>
  <TR><TD>Status</TD><TD >Ready</TD></TR>
  <FORM ACTION="/sas5/tasks_servlet»
   <TABLE>
    <TR><TD><INPUT TYPE="submit"VALUE="Resend"
    NAME="btnresendForm»</TD></TR>
  </TABLE>
  </FORM>
 </TABLE>
 <TABLE>
  <TR><TD>
  <A HREF="/sas5/log_servlet/searchParameters">Parameters</A>
  <A HREF="/sas5/log_servlet/searchIOLog">IO Log</A>
  <A HREF="/sas5/log_servlet/searchMMLLog">MML Log</A>
  <A HREF="/sas5/log_servlet/searchErrorLog">Error Log</A>
  </TD></TR>
 </TABLE>
</HTML>
```

Figure 3.3: A web page snippet of Figure 3.4. The link information is set in bold face type.

The user interface consists of web pages that are hierarchically connected to each other. The pages are connected with a link so that adjacent pages can be reached by clicking a link. Therefore, the user interface can be considered as a tree that branches every time the user navigates further in the application.

The hierarchical structure of the user interface is acquired by starting the link parsing from the main page and following the links to the child pages until there is no child pages left. The information of adjacent pages is stored with the link information so that they form a hierarchical presentation of the UI.

Acquiring the UI structure is possible to automate. Starting from the main page, the search can be executed exhaustively by parsing the anchor tags from the HTML

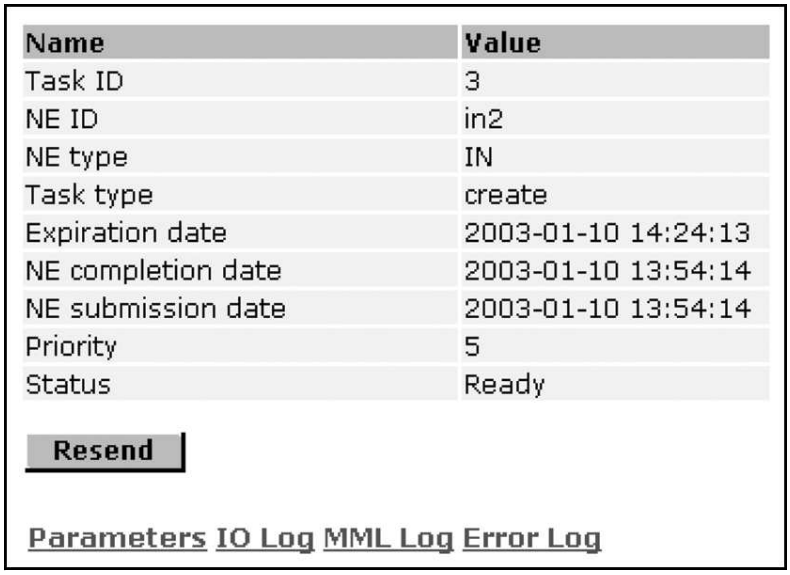

Figure 3.4: The page is stylised with Cascading Style Sheet (CCS) but the structure is similar than in pure HTML.

description and sending a new HTTP request with each anchor element's target text to the web server. During the search, the information of adjacent pages is collected. If some link points to an already requested page, the request is not sent but the link information is stored.

This is an efficient method for collecting the UI structure if the pages are not branching to external applications. It is easy to understand that if there is one link pointing to an external page, it might explode the whole search.

3.2.3 Processing the Dynamic Content

Collecting and transforming the static UI pages is straightforward, because the weblog can be compared to the user interface description. In that case, only the information of interest can be filtered for analyses. If a user interface page contains anchor elements whose contents change in time, it will cause discrepancy between the user interface description and the logged user actions. The changing UI description causes the problem.

There are two solutions to this problem. The first one is to update the description every time the content of the UI changes. This is a heavy operation and it should

be avoided, especially if the dynamic content changes frequently. The second solution is hidden in the concept of the list.

Information in a list has the same kind of information structure. In other words, list entries have the same information frame in where only the information makes each entry individual. By abstracting the information and remaining only the frame, the entries can be handled as a whole. Notice the difference between the use of the term abstraction here and in Chapter 2.4.2.

Following the idea, different dynamic contents can be handled, if there is some regularity in their structure. Depending on the level of dynamics on a page, abstraction causes different amount of information loss in the collection. If the whole structure of a page is dynamic, the abstraction misses all information that is more detailed than “a page”. But if the page’s dynamic is on the level of lists, then only the list content is abstracted. In the scope of this thesis, only the list structures are handled.

Depending on the application, navigation activities to single list entries have different levels of interest. Single entries are meaningful only in special cases and they cannot be recognized from the others. Conversely, the list entries as a whole represent an interesting part of the user interface. The user actions relating to them provide a good insight to the user navigation in the application.

Here is an example of the query parts of URIs from the weblog. The parameter values are different, while the structure is the same:

```
GET /sas5/network_servlet/showConnectionDetails?conId=
  mds1_hlr2&sourceNe=mds1&targetNe=hlr2 HTTP/1.1
GET /sas5/network_servlet/showConnectionDetails?conId=
  sms5_sms4&sourceNe=sms5&targetNe=sms4 HTTP/1.1
```

After abstraction, these entries are the following:

```
GET /sas5/network_servlet/showConnectionDetails?conId=
  FFF&sourceNe=FFF&targetNe=FFF HTTP/1.1
GET /sas5/network_servlet/showConnectionDetails?conId=
  FFF&sourceNe=FFF&targetNe=FFF HTTP/1.1
```

The result is that both entries are counted as the same. They loose their meaning as single elements and come meaningful only as instances of an abstracted class of the entries.

3.3 Design Model for Analyses

There is no single rule how the analysis should be constructed. Each application has features, which require special analysis to thoroughly study their nature. This thesis provides a general analysis methodology that can be used in web-based applications in general. The methodology is a combination of the actual use logging approaches as described in Chapter 2.4.

In the end, the analyses do not provide straight answers to the development of the user interface. Neither do they give straight answers to the usability of an application. Merely, they are like tools for the interface designers to better understand the user's behaviour in the application. All analyses need an interpretation from a person who knows the application and the user interface before they can be fully utilized.

The selection of analysis methods is based more on intuition than on any describable method. As said above, the analyses are like tools and they have a special area in which they work in the best way. The provided methodology is a toolbox to which a purposeful set of tools was selected so that the box is useful in different kinds of situations.

3.3.1 Analyses Scenario

At first, the UI designer needs a good overview of the application usage. He wants to have a clear picture of the parts that the users are navigating the most. Based on that information, the development can be focused to the most frequently used sections of the user interface in where the improvements have the best effect. This need can be fulfilled with the Visualisation of User Selection analysis, which is presented in the next chapter.

When the designer has a good view of the user navigation, the next step is to analyse the particular interests of the user. The focus is set on the single actions, or a sequence of them. When the user is browsing the interface, he is usually performing a task. A way to study the frequency of tasks is to follow the frequency of different user action sequences. By varying the length of analysed sequences, the tasks can be seen through the noise of unmeaningful actions. This analysis is called User Action Sequence Frequencies.

The last analysis is for finding the dependencies of the user interface sections. The tasks might have an order so that certain tasks are following each other. This analysis gives insight to the diffusion of the tasks in the user interface. If the diffusion is great, the user changes the user interface section often and does not do many selections in one section. This analysis is called User Interface Transition Steps with Measured Probabilities.

3.3.2 Visualisation of User Selection

The idea of this analysis is to show the distribution of user selections graphically on a user interface page. In short, the user navigation activities are collected and they are shown with column presentations on top of the actual user interface page to which they belong.

This analysis gives a good insight into the distribution of user selections on e.g. menu pages, which are the branching points of the user interface. The basic functionality is to show the user clicks on a single page, but also the clicks on the underlying pages of a single link can be added to the graph. By doing so, the total number of user activities in the different application sections are easy to understand.

The strength of this analysis it that it provides a quick view to the user interface sections in which the users are navigating the most. The graph is easy to understand and it can be shown to people who do not necessarily know much about the application or even the graphical user interfaces.

The graph of user selection analysis applies to the counts and summary statistics (Section 2.4.3) and visualisation (Section 2.4.7) approaches. In this thesis, the vi-

sualisation is done manually by using external programs. Automated visualisation is not in the scope of this thesis. However, the results can be used as an input to some external software that handles the visualisation.

3.3.3 User Action Sequence Frequencies

The user's navigation paths provide a good view on the activities that the user has to perform before reaching a certain, desirable, action. If some frequently used action requires a long navigation path, it could be moved closer to the main page. This analysis gives visibility to the paths that the user is following.

This analysis applies to the abstraction (see Section 2.4.2) and the counts and summary (see Section 2.4.3) approaches. The user activities are abstracted to be meaningful only as members of sequences. After they are abstracted, the counts and summary methods are applied to count the frequency of different sequences. When the analysis is executed, the sequences are disassembled and single activities can be studied.

3.3.4 User Interface Transition Steps with Measured Probabilities

The user interface can be divided into segments that are relevant with the tasks that the application was built for. Accordingly, the segments also represent the parts of the user interface in which the user does a complete task or a subtask. These parts can be chosen according to different premises but a good starting point is to choose the sections basing to the main menu structure.

In order to divide the collected user actions to different segments, the data has to be transferred first (see Section 2.4.2). The menu categories form the abstraction levels. Each menu category forms its own abstraction level so that all underlying user interface elements of a certain menu item are first abstracted and then recoded to be the same. This abstraction is relevant if the user interface does not have feedback links among the menu trees.

The user interface segments form the *states* of the user interface. The user's nav-

igation enters him either to a new state or remains him in the same state. Transitions between these states are counted and their probabilities are calculated. Even though the idea of handling the user interface as a state machine is rather general, at least Guzdial (1996) has introduced it in his study.

3.4 Implementation of Task Frequency Tool

TFT was implemented to be as open as possible. It is a stand-alone software, which means that it is not dependent of any external software products. The idea was to develope a package that can easily be extended if needed. For example, the collection and the analysis parts of TFT are extendable with other implementations or vice versa. The source code is provided in Appendix A.

3.4.1 Software Platform

In order to provide hardware independent implementation, it was decided to program TFT with Java. In the beginning of the implementation, the used Java version was 1.3.1. After finding out that 1.3.1 does not support regular expressions and a few other useful features, the Java version was updated to 1.4.1.

3.4.2 Software Architecture

The class selection is based on the entities that are involved in TFT. The class diagram of TFT is depicted in Figure 3.5.

According to a good Java programming practice, the source code is commented in *javadoc*. So, the code contains an in-depth documentation of the classes and their methods. Only a short description of the classes is provided in the following:

UIDescrHandler is built around the user interface description file. By separating the UI description to a file, an external collector can be used for collecting the elements to a file (see Section 3.2.2). All UI elements are described in a file from which this class reads and provides them to other classes. The hierarchy between

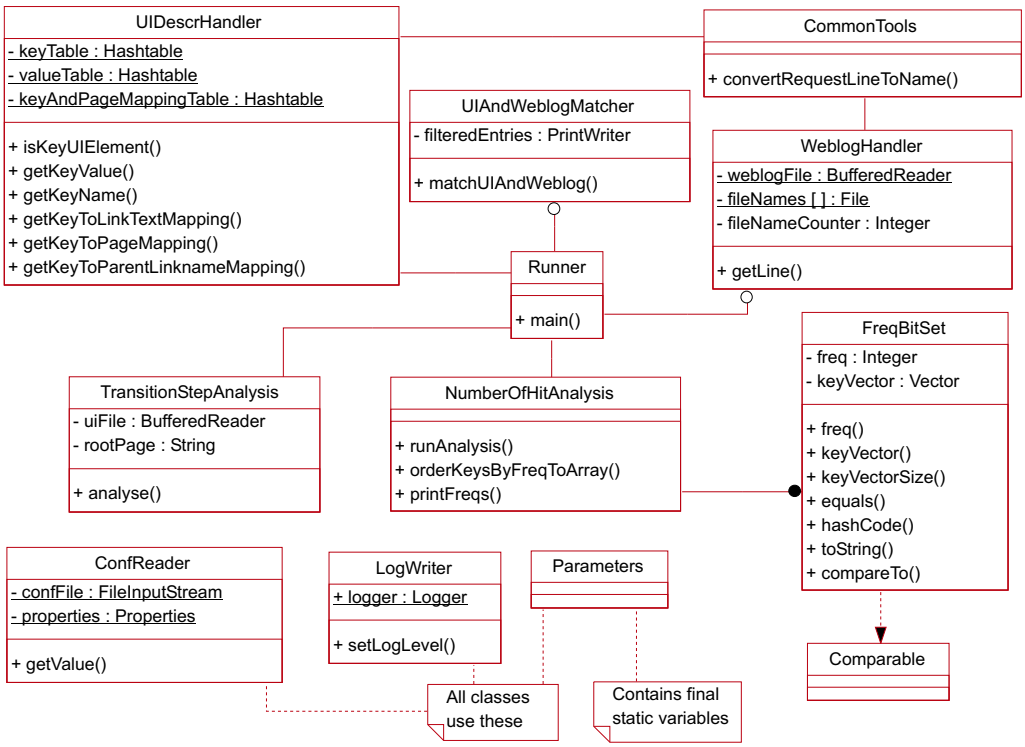

Figure 3.5: The class diagram of TFT.

the elements is also included to the file. The syntax of the description file is the following:

```
link -> {
  query separator1 page separator1 separator2 link separator2
}
```

query Query part of the HTTP request

separator1 Page name separator (e.g. #)

page Name of the page in which the element is located

separator2 Link text separator (e.g. ”)

link The link text in the user interface

The separators, description file name and the description file parsing regular expression are configurable through the configuration file (see Section 3.4.3). If there

is dynamic content in the user interface, they have to be written in angle brackets. There is an example of an actual description file in Appendix B.

WeblogHandler takes care of the weblog file that was collected in the web server. It provides an interface to the weblog file that the other classes can use. A wide variety of weblog formats can be used, because this class reads necessary information from the weblog with a regular expression that is read from the configuration file. For example, activities from a certain host can be analysed by giving the host's IP address in the configuration file. Also, the name of the weblog file is read from the configuration file.

UIAndWeblogMatcher is a filter class that reduces the noise from the collected weblog. It compares the weblog entries to the UI description through *UIDescrHandler* class and writes the filtered user actions to the file. The entries in the file are only the intentional user actions that generated a HTTP request to the server. The entries are in a "key"-format, which means that the dynamic content of an entry is abstracted to a static form. That is the same format in which *UIDescrHandler* internally handles the entries.

TransitionStepAnalysis counts the transitions between the different segments - *states* - in the UI. The class takes the first level of the menu structure as the base of the segmentation. Then, each user action is abstracted and recoded to belong to one of the states. Finally, the transitions between two adjacent user actions are counted and they are outputted.

LogWriter wraps the Java API's *Logger* class. Actually, *LogWriter* is a convenience class for providing a static reference to the *Logger*.

Runner contains the main method and pulls all classes together. If there are some new analysis implementations, this class should be extended by writing appropriate calls to new analysis classes.

Parameters class provides the "hard-coded" values. They are e.g. the name of the configuration file and the parameter names in it.

NumberOfHitAnalysis is an analysis class. The class counts the number of hits of each user interface element and provides them on the screen. By giving a value to 'seqlgth' variable in the configuration file, the length of the user action sequence

can be set. If the value is '1', the class counts the number of hits to each single user interface element. Otherwise, the user actions are handled as sequences and the number of each sequence is provided.

FreqBitSet implements Java API's *Comparable* class. The purpose of *FreqBitSet* is to provide a sortable data structure element for the analysis classes.

CommonTools is a storage class for tool methods that are common for several classes.

ConfReader supplies methods for other classes to read values from the configuration file. Actually, *ConfReader* uses Java API's *Properties* class, which provides a convenient way to read configuration values from a file.

3.4.3 Configuration

TFT is configured through a text file, whose name is *tft.properties*. Below, there is an example of the file:

```
tftlogfile        ~/tft/tft.log
weblogdir         ~/logs/localhost_access\_log.2002-09-06.txt
weblogregexp      ^10\.20\.20\.212.*GET[ ](.*)[ ]HTTP.*
dbglevel          FINE
seqlgth           3
uidscrpt          ~/tft/uielements.txt
uidscrptregexp    (.*) #.*LINK
uielemfile        useracts.txt
pagenamedelimiter \#
linknamedelimiter ''
rootpage          Main
```

tftlogfile Name of the TFT's log file

weblogfile Name of the weblog file

weblogregexp Regular expression for parsing the wanted lines from the weblog

dbglevel Debug level of TFT. One of the following: SEVERE, WARNING, INFO, CONFIG, FINE, FINER, FINEST

seqlgth Length of the analysed user action sequences

uidescription Name of the UI description file

uidescriptionregexp Regular expression for parsing the link lines from the UI description

uielemfile Name of the filtered user action file to which *UIAndWeblogMatcher* class writes them

pagenamedelimiter Delimiter for page names in the UI description file

linknamedelimiter Delimiter for link names in the UI description file

rootpage The name of the root page in the UI description file

3.4.4 Use of Task Frequency Tool

Using TFT is rather simple. After setting the basic configuration to the configuration file, the user interface description file and the collected weblog files have to be be configured to the system. The UI description file location, the regular expression for parsing, page and link name delimiters have to be set to appropriate values. Finally, the weblog file location and the regular expression for parsing it have to be set.

When the configuration is done, TFT analyses can be executed by running Java virtual machine with the following command:

```
/usr/tools/java/bin/java Runner
```

If the size of the weblog files is great, the analysis might take some time to finish. In that case, the progress of analysis can be followed from the TFT log file. When the execution is ready, TFT prints the results on the screen.

Every time TFT is executed, it checks if the filtered user actions file already exists. Because filtering takes some time, it is not necessary to do if the weblog data has

not changed. Therefore, it is up to the user of TFT if he wants to delete the user action file and do filtering from the scratch.

Chapter 4

Subscriber Administration System

This chapter provides an introduction to the application environment, which is in focus in the case study of this thesis. The first section describes what a mediator software is and what its position in telecommunications network is. After that, a real mediator software, MDS/SAS 5.0, is introduced.

4.1 Mediator Software in Telecommunications Network

Mediator is a system through which a telecommunications network operator can manage subscriber information in telecommunications network. Once the mediator is configured to the operator's system, it hides the underlying network and provides an interface for a Business Support System (BSS) to send service requests to. Therefore, the operators do not have to have profound knowledge about the actual network. The operator can create, modify and delete subscribers and perform data queries and service provisionings to them.

The basic model of MDS/SAS is simple. The following example illustrates how a mediator is related to the telecommunications world:

> Suzy, an active mobile phone user, has just heard of a new campaign of PhoneWiz, which is a local teleoperator company. By contracting

them now, 50 euros worth of call time and a Homeline service is provided for free! This campaign she cannot resist.

Suzy walks eagerly to the branch office of PhoneWiz where Liz, an energetic customer servicer, is welcoming her. Liz declares all conditions to Suzy and because of her good mood, she includes VoiceMail service to the contract for free! Suzy thinks Liz is very cool :).

Before Suzy's new services can be used, Liz inserts new customer information to PhoneWiz's BSS. Network element specific data is sent to the mediator, which transforms it into a specific element dependent commands and propagates them to the right elements. After the services, e.g. phonering, are created, an acknowledgement is sent to the mediator, which forwards it to BSS.

Suzy has just left the branch office and walks towards a university in where her biology class will begin in 20 minutes. Suddenly, her phone beeps in her pocket. It is a welcoming text message from PhoneWiz. Suzy feels happy that changing the operator was so easy and quick. She decides to call her boyfriend right after the class because now it's free, for the first 50 euros.

The mediator software can be thought to be a gate between the high-level semantic and low-level syntactic data as depicted in Figure 4.1. The software receives service requests, which BSS passes to it. They can be mapped to people's needs in their everyday life, e.g. "I keep missing my phonecalls all the time. I want that people can leave me messages if they do not reach me". The mediator disassembles the service requests to low-level segments according to the configured rules. The segments have a meaning only when they exist together. If one of the segments is lost, the whole service request is lost.

4.2 MDS/SAS 5.0

MDS/SAS 5.0 is a web-based software, which means that it is used through a web browser. The software itself is independent of the graphical user interface. Only the configuration and system status check is done from the web browser. In the

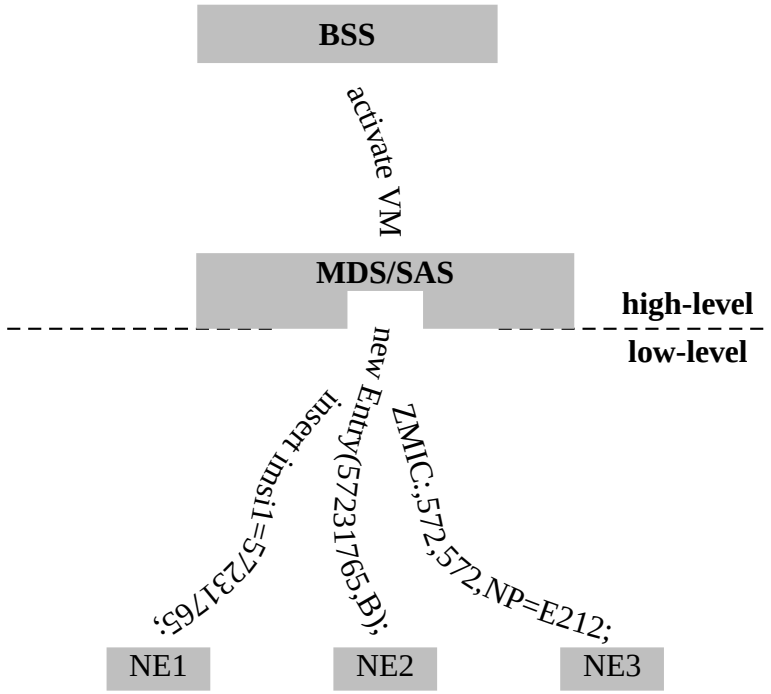

Figure 4.1: The mediator functions as a gate between the high- and low-level data in telecommunications network. The service requests exist on high-level and they are disassembled into abstract segments in the mediator. The abstract segments are network element specific commands.

optimal case, MDS/SAS is configured to the operator's system only once. After the configuration, the software is used only for checking that everything is working correctly and that no errors are reported.

4.2.1 Graphical User Interface

The GUI contains three different parts as can be seen in Figure 4.2. The top row (1) contains links to the main functions and it always remains the same. According to the chosen main function, the left column provides a different set of links. When the user selects a link in the left column (2), the middle part (3) is activated. According to the chosen link, information and further options are shown in the middle part of the screen.

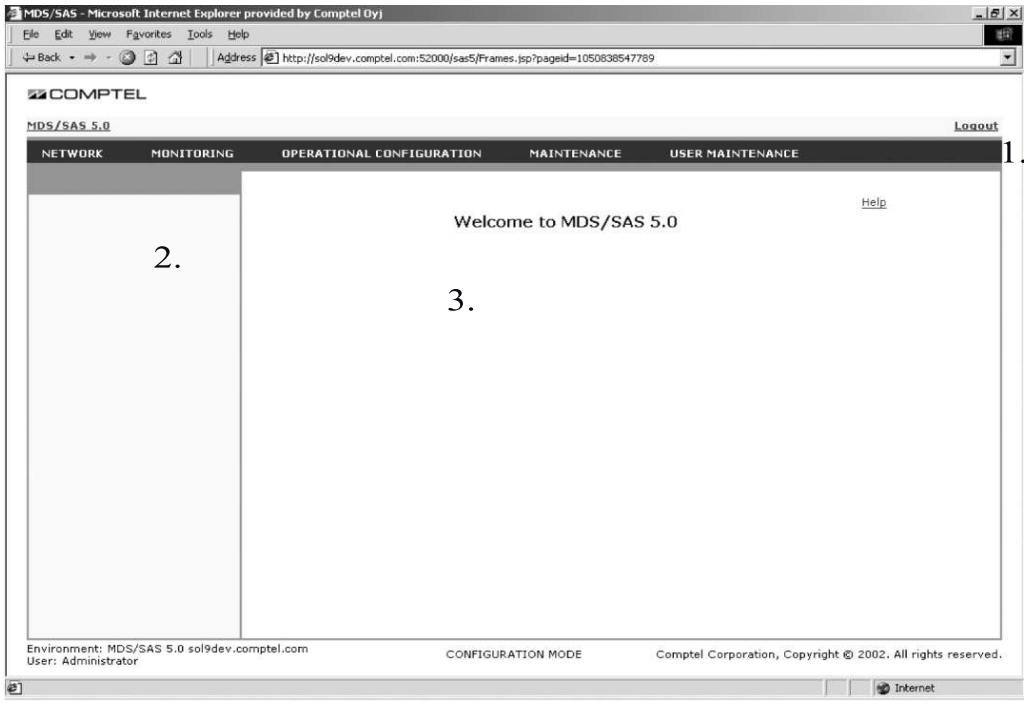

Figure 4.2: The main page of MDS/SAS 5.0. The main menus are on the top (1). The left part of the page (2) is reserved for the sub-menus that are activated after the main menu is selected. After the user selects a link in the left part, the middle part (3) activates.

The first two iterations of the wanted actions take place in the top and left parts of the GUI. Then they continue in the middle part. The deepest iteration path contains six levels. In simpler terms, the first two iterations fix the appearance of the top

and left parts of the GUI while the following iterations take place in the middle part. An illustration of the GUI link structure is presented in Figure 4.3. The user starts browsing the user interface from the Main page and continues to a desired page via appropriate sub-menu links, which are Network, Monitoring, Operational Configuration, Maintenance and User Maintenance. In addition to the sub-menu links, there are three other links available on the main page, but they do not provide underlying link structures.

Although the most parts of the GUI are static, there are a few pages that contain some dynamic structures. They are the list pages where the number of list items depends on the mediated requests from the BSS to the network elements. Also, some list pages contain the initially configured teleoperator's network elements, whose number naturally varies between the operators.

4.2.2 Users

The users of MDS/SAS can be divided into two groups. The first group of users are the employees of Comptel, who work at the customer site during the software installation phase. In brief, they configure MDS/SAS to the customer's network elements and define how the service requests are disassembled to network element specific commands. In addition, they do some basic testing for ensuring the correct functioning of configurations.

The second group of users are the people in customer organizations who are in charge of operating the system while it is in production. The operators' task is primarily to follow the error messages, or desirably, the absence of them. They also configure MDS/SAS if the underlying network infrastructure changes after the initial configuration.

There is a difference in MDS/SAS use experience between the user groups. Comptel's employees, who are working at the customer site, have a long time experience in using the software whereas the customers' users are usually dealing with the software for the first time. It is probable that the customers' users have technical background.

Even though MDS/SAS is a special software focused to a limited user group, the

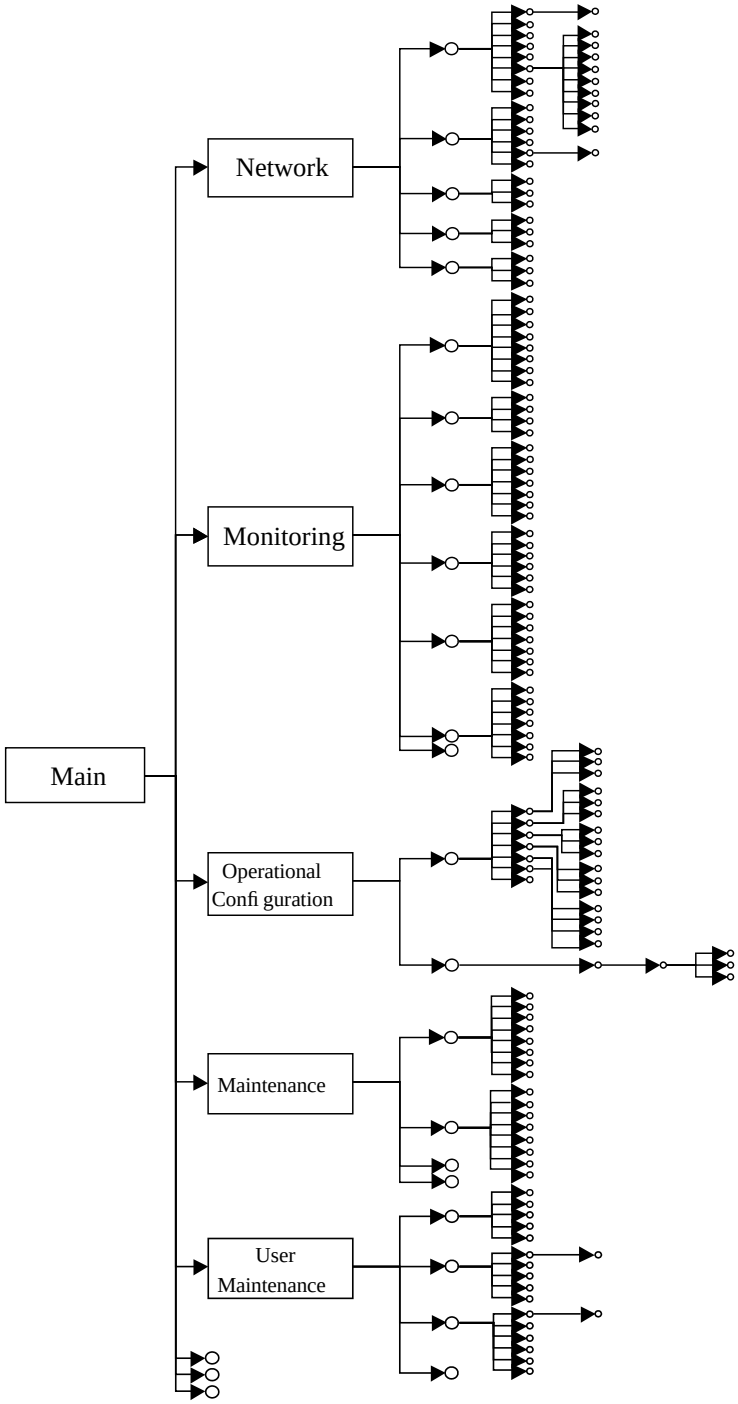

Figure 4.3: Hierarchical GUI description. The user starts browsing MDS/SAS 5.0 from the Main page and continues to a desired page via one of the five sub-menu links. Each circle element represents a link in the GUI. The end of a navigation path is a link that points to a page that does not have any further links.

clarity of the UI is important. An incorrect system configuration could result to interruptions in telephone connections or overloads and system crashes in telecommunications networks. Usually incorrect configurations also need a contact to the help desk of Comptel for instructions. Because of the great number of installations around the world, the help desk could get overloaded and customer services get congested, which would escalate the customers' problems even more.

Chapter 5

Applying Task Frequency Tool for MDS/SAS 5.0

This chapter describes the case study that I conducted for MDS/SAS 5.0. The goal was to test the model and the implementation in a real life situation. First, the test environment is described. Then, the collected data is presented and the results of TFT analyses are reported. According to the policy of Comptel, only part of the results of the analyses are reported in this thesis.

5.1 Collection Environment

Prerequesites for the test environment limited the number of Comptel's customers whose MDS/SAS 5.0 installations could have been used in this study. First, they had to be running specifically the software version 5.0 amongst several other versions. Second, they had to be rather big operators because of the volume of AUL data; I wanted it to be as great as possible. Also, one requirement was that the system should be in production use and not in test use in order to collect authentic data from real use situation.

After I weighted the conditions above with the people in Comptel, I chose one customer to approach. I contacted the customer through a project manager, who already had contacts to the company. The actual approach was a proposal letter

that the project manager sent to the company.

The customer was concerned for their privacy and did not want their name to be published in this thesis. Furthermore, they were worried that the analyses could reveal something of their network infrastructure. Because of the privacy concerns, the company is nicknamed as PhoneWiz hereafter. The network infrastructure issue was unnecessary because the detailed information is abstracted before the analyses. Therefore, there is no possibility that network specific information is revealed in analyses.

There was no previous information of the active use of MDS/SAS after it is configured to the teleoperator's system. For that reason, the data collection time was left open and only a rough approximation from a week to a month could be given to PhoneWiz

5.2 Configuring TFT for MDS/SAS 5.0

In the case of MDS/SAS, the UI elements were collected manually. I did not want to use extra time to implement an automated UI element collection module to TFT. So, I collected the link element information by browsing the UI and writing a hierarchical document of the link structure in MDS/SAS. There is an example of the collected information in below:

```
Main -> {
  /sas5/naviServlet #Main Menu# "MDS/SAS 5.0"
  /sas5/index.jsp #Main Menu# "Logout"
  /sas5/naviServlet/showNetwork #Main Menu# "Network"
  /sas5/naviServlet/showMonitoring #Main Menu# "Monitoring"
  /sas5/naviServlet/showOperConf #Main Menu# "Oper_Conf"
  /sas5/naviServlet/showMaintenance #Main Menu# "Maintenance"
  /sas5/naviServlet/showUserMaint #Main Menu# "User_Maint"
}

Network -> {
  /sas5/networkServlet #Network menu# "Network_Model"
  /sas5/neTypesServlet #Network menu# "NE_Types"
  /sas5/neIfTypesServlet #Network menu# "NE_Interface_Types"
  /sas5/connectionTypesServlet #Network menu# "Connection_Types"
```

```
  /sas5/defNeParamsServlet #Network menu#
        "Default_NE_Parameters"
}

NE_Types -> {
  /sas5/ne_types_servlet/showNETypeDetailsReadOnly?id=<NE Type>
        #NE Types submenu# "NE_Type_Details"
  /sas5/ne_types_servlet/showNETypeDetails?id=<NE Type>
        #NE Types submenu# "Modify_NE_Type"
  /sas5/ne_types_servlet/showNETypeDeleteConf?id=<parameter>
        #NE Types submenu# "Delete_Parameter"
  /sas5/ne_types_servlet/createNEType
        #NE Types submenu# "Add_New_NE_Types"
}
```

The UI description file contains 39 link sets that describe the link structure of the application. Altogether, there are 183 abstracted links in the file. The complete list of the links is presented in Appendix B.

5.3 Collected Data

There was no experience of how much time it would take to collect enough data to get some significant amount of actual use logging events. Therefore, there was only an approximation that user actions should be counted in thousands or even in tens of thousands to get reliable results from the analyses.

The accumulation of the collected weblog was followed weekly. After the first two weeks, it was clear that the approximated maximum time limit of collection would exceed because of the low use activity of MDS/SAS. Consequently, the collection continued for eight weeks until it had to be stopped to meet the time limits of this thesis. Otherwise, it would have been continued to get more data for the analyses.

An example of collected weblog data is in below (IP-address is shortened for security and timestamp and number of sent bytes for brevity). The whole weblog from the same session is in Appendix C.

```
x.x.x.160--[13/3/2003:12:43:41] "GET /sas5/index.jsp HTTP/1.1" 200
x.x.x.160--[13/3/2003:12:43:41] "GET /sas5/styles/sas5_ie.css HTTP/1.1" 200
```

```
x.x.x.160--[13/3/2003:12:43:41] "GET /sas5/images/dot.gif HTTP/1.1" 200
x.x.x.160--[13/3/2003:12:43:41] "GET /sas5/images/comptel.gif HTTP/1.1" 200
x.x.x.160--[13/3/2003:12:43:46] "POST /sas5/ui_user_servlet HTTP/1.1" 302
x.x.x.160--[13/3/2003:12:43:46] "GET /sas5/navigation_servlet/show?pageid=
                                 1047552226441 HTTP/1.1" 302
x.x.x.160--[13/3/2003:12:43:46] "GET /sas5/Frames.jsp?pageid=
                                 1047552226459 HTTP/1.1" 200
x.x.x.160--[13/3/2003:12:43:46] "GET /sas5/Header.jsp HTTP/1.1" 200
x.x.x.160--[13/3/2003:12:43:46] "GET /sas5/Menu.jsp HTTP/1.1" 200
x.x.x.160--[13/3/2003:12:43:46] "GET /sas5/EmptyMenu.jsp HTTP/1.1" 200
x.x.x.160--[13/3/2003:12:43:46] "GET /sas5/Empty.html HTTP/1.1" 200
x.x.x.160--[13/3/2003:12:43:46] "GET /sas5/Welcome.jsp HTTP/1.1" 200
x.x.x.160--[13/3/2003:12:43:46] "GET /sas5/Footer.jsp HTTP/1.1" 200
x.x.x.160--[13/3/2003:12:43:46] "GET /sas5/images/dot.gif HTTP/1.1" 200
x.x.x.160--[13/3/2003:12:43:46] "GET /sas5/styles/sas5_ie.css HTTP/1.1" 200
x.x.x.160--[13/3/2003:12:43:46] "GET /sas5/images/comptel.gif HTTP/1.1" 200
x.x.x.160--[13/3/2003:12:43:46] "GET /sas5/images/menuBG.gif HTTP/1.1" 200
x.x.x.160--[13/3/2003:12:43:48] "GET /sas5/navigation_servlet/showEmpty-
                                 Network HTTP/1.1" 302
x.x.x.160--[13/3/2003:12:43:48] "GET /sas5/Frames.jsp?pageid=1047552228058
                                 HTTP/1.1" 200
x.x.x.160--[13/3/2003:12:43:48] "GET /sas5/Header.jsp HTTP/1.1" 200
x.x.x.160--[13/3/2003:12:43:48] "GET /sas5/NetworkNavigation.jsp
                                 HTTP/1.1" 200
x.x.x.160--[13/3/2003:12:43:48] "GET /sas5/NetworkMenu.jsp HTTP/1.1" 200
x.x.x.160--[13/3/2003:12:43:48] "GET /sas5/EmptyNetwork.jsp HTTP/1.1" 200
x.x.x.160--[13/3/2003:12:43:48] "GET /sas5/Empty.html HTTP/1.1" 200
x.x.x.160--[13/3/2003:12:43:48] "GET /sas5/Footer.jsp HTTP/1.1" 200
x.x.x.160--[13/3/2003:12:43:48] "GET /sas5/images/dot.gif HTTP/1.1" 200
x.x.x.160--[13/3/2003:12:43:48] "GET /sas5/images/comptel.gif HTTP/1.1" 200
x.x.x.160--[13/3/2003:12:43:48] "GET /sas5/styles/sas5_ie.css HTTP/1.1" 200
x.x.x.160--[13/3/2003:12:43:48] "GET /sas5/images/menuBG.gif HTTP/1.1" 200
x.x.x.160--[13/3/2003:12:43:53] "GET /sas5/network_servlet HTTP/1.1" 302
```

After filtering the “noise” from above, the intentional user actions are the following:

```
/sas5/navigation_servlet/showEmptyNetwork
/sas5/network_servlet
```

In the end, the size of the weblog was 4,6 MB and it contained more than 44000 entries. Reducing the noise and filtering only the intentional user actions reduced the size of the useful log file to contain only a bit more than 2100 actions. So, the ratio between the noise and the meaningful user actions was about 5%. The rest of the weblog consisted of HTTP requests for e.g. pictures and style sheet files.

5.4 Analyses

The analyses were conducted for the collected and filtered actual use logging events. The analyses are described in Section 3.3. TFT provides only numerical information of the analyses. To illustrate the results here, they are described with graphical presentation. The illustrations are a graphical transformation of the results of the analyses.

5.4.1 Visualisation of User Selection

TFT provides hit-analysis information for the single user interface elements. It counts the single user selections for each element and ranks them in descending order. After that, it outputs the frequency of the user selection with the name of the page and the link text. Therefore, information is easy to map into the user interface and different data representations can be applied.

There was 44 abstracted user interface elements that were used during the test period. In other words, 24 % of the UI elements were used at least once.

The following lines are a selection of the original output of TFT. They describe the usage of the links in the top part of the user interface. It would be helpful to refer to Appendix B to understand the ’Page’ and ’link text’ variable.

```
Frequency :’272’
Page :’Main Menu’ link text :’Monitoring’
Frequency :’126’
Page :’Main Menu’ link text :’Logout’
Frequency :’64’
Page :’Main Menu’ link text :’Network’
Frequency :’32’
Page :’Main Menu’ link text :’Maintenance’
Frequency :’16’
Page :’Main Menu’ link text :’Operational Configuration’
Frequency :’4’
Page :’Main Menu’ link text :’User Maintenance’
```

To illustrate the results better, I leaned the top part of the MDS/SAS 5.0’s user interface page and drew a graphical presentation of the user selections on top of it.

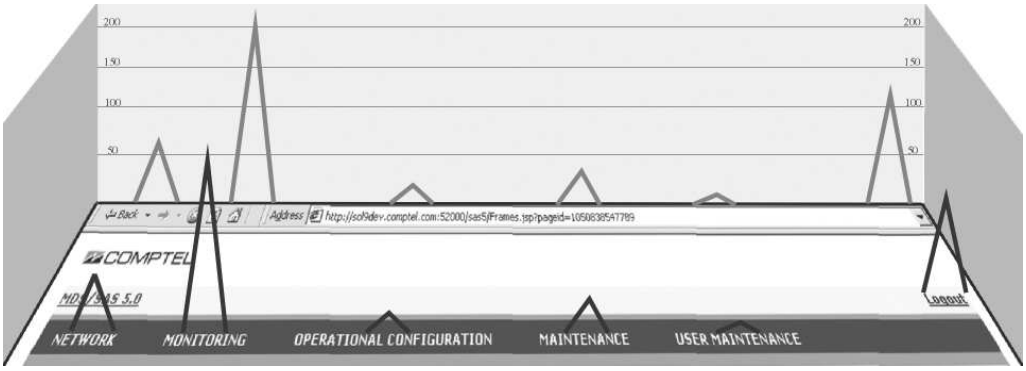

Figure 5.1: Visualisation of user selections on the user interface's top panel. The user selections are presentented graphically with different sized peaks on top of the user interface page. The higher the peak, the more user selections that particular element has had during the test period. For those interested, the figure was prepared with CorelDRAW 10.

From the graphical illustration, it is easy to get an idea of the distribution of the user selections on the top panel. The most used user selections were Monitoring and Logout links. It is also easy to see that Operational Configuration and User Maintenance links were selected very rarely. A surprising factor is that the main page contains the link 'MDS/SAS 5.0' that was never selected. TFT registers only the links that were selected at least once and therefore, the link 'MDS/SAS 5.0' is not listed in the previously described output.

5.4.2 User Action Sequence Frequencies

In order to study the different navigation paths of the user, information regarding the most frequent user selection combinations is important. TFT provides a possibility to study the different length combinations and find out the most frequently occurring sequences.

MDS/SAS 5.0 was analysed with sequence lengths from two to five. When the length was increased to over five, the occurrences of sequences drop to less than ten. For this purpose, it was relevant to limit the upper limit, because of the reliability of the results. Also, some unrelevant sequences were removed from the results. They contained e.g. Logout selection between the first and the last selections.

The results are reported so that the frequency of the sequence is on the first line and the sequence itself is after that. To better understand the results here, the MDS/SAS 5.0's link structure should be read from Appendix B. The occurred sequences and their frequencies are reported below:

```
Sequence length: 2
Frequency :'83'
Page :'Main Menu' link :'Monitoring'
Page :'Monitoring menu' link :'Requests'
Frequency :'79'
Page :'Requests submenu' link :'Request_Details'
Page :'Request Details <ID> subsubmenu' link :'Task_ID_Number'
Frequency :'60'
Page :'Tasks submenu' link :'Task_Details'
Page :'Tasks submenu' link :'Task_Details'
Frequency :'49'
Page :'Maintenance menu' link :'System_Status'
Page :'Maintenance menu' link :'System_Status'
Frequency :'47'
Page :'Network Model submenu' link :'NE_Details'
Page :'Network Model submenu' link :'NE_Details'
Frequency :'32'
Page :'Monitoring menu' link :'Requests'
Page :'Requests submenu' link :'Request_Details'
Frequency :'31'
Page :'Request Details <ID> subsubmenu' link :'Task_ID_Number'
Page :'Task Details subsubmenu' link :'IO_Log'


Sequence length: 3
Frequency :'31'
Page :'Tasks submenu' link :'Task_Details'
Page :'Tasks submenu' link :'Task_Details'
Page :'Tasks submenu' link :'Task_Details'
Frequency :'27'
Page :'Maintenance menu' link :'System_Status'
Page :'Maintenance menu' link :'System_Status'
Page :'Maintenance menu' link :'System_Status'
Frequency :'27'
Page :'Requests submenu' link :'Request_Details'
Page :'Request Details <ID> subsubmenu' link :'Task_ID_Number'
Page :'Task Details subsubmenu' link :'MML_Log'
Frequency :'26'
Page :'Network Model submenu' link :'NE_Details'
Page :'Network Model submenu' link :'NE_Details'
Page :'Network Model submenu' link :'NE_Details'
Frequency :'21'
Page :'Requests submenu' link :'Request_Details'
Page :'Request Details <ID> subsubmenu' link :'Task_ID_Number'
Page :'Task Details subsubmenu' link :'IO_Log'
Frequency :'15'
Page :'Main Menu' link :'Monitoring'
Page :'Monitoring menu' link :'Requests'
```

```
Page :'Requests submenu' link :'Request_Details'

Sequence length: 4
Frequency :'20'
Page :'Network Model submenu' link :'NE_Details'
Page :'Network Model submenu' link :'NE_Details'
Page :'Network Model submenu' link :'NE_Details'
Page :'Network Model submenu' link :'NE_Details'
Frequency :'17'
Page :'Tasks submenu' link :'Task_Details'
Page :'Tasks submenu' link :'Task_Details'
Page :'Tasks submenu' link :'Task_Details'
Page :'Tasks submenu' link :'Task_Details'
Frequency :'15'
Page :'Maintenance menu' link :'System_Status'
Page :'Maintenance menu' link :'System_Status'
Page :'Maintenance menu' link :'System_Status'
Page :'Maintenance menu' link :'System_Status'
Frequency :'8'
Page :'Requests submenu' link :'Request_Details'
Page :'Request Details <ID> subsubmenu' link :'Task_ID_Number'
Page :'Task Details subsubmenu' link :'MML_Log'
Page :'Requests submenu' link :'Request_Details'

Sequence length: 5
Frequency :'14'
Page :'Network Model submenu' link :'NE_Details'
Page :'Network Model submenu' link :'NE_Details'
Page :'Network Model submenu' link :'NE_Details'
Page :'Network Model submenu' link :'NE_Details'
Page :'Network Model submenu' link :'NE_Details'
Frequency :'12'
Page :'Maintenance menu' link :'System_Status'
Page :'Maintenance menu' link :'System_Status'
Page :'Maintenance menu' link :'System_Status'
Page :'Maintenance menu' link :'System_Status'
Page :'Maintenance menu' link :'System_Status'
Frequency :'9'
Page :'Tasks submenu' link :'Task_Details'
Page :'Tasks submenu' link :'Task_Details'
Page :'Tasks submenu' link :'Task_Details'
Page :'Tasks submenu' link :'Task_Details'
Page :'Tasks submenu' link :'Task_Details'
Frequency :'8'
Page :'Request Details <ID> subsubmenu' link :'Task_ID_Number'
Page :'Task Details subsubmenu' link :'MML_Log'
Page :'Requests submenu' link :'Request_Details'
Page :'Request Details <ID> subsubmenu' link :'Task_ID_Number'
Page :'Task Details subsubmenu' link :'MML_Log'
```

The shortest sequence length shows the navigation path segments that the user is following. When the sequence length is increased, the bigger picture of the system is easy to see. Based on the results, it seems that the user is monitoring the system.

If we analyse one of the most frequently occurring sequences, we notice that the user browses from the Main page to the Monitoring page to see request and task details. When the user has entered the task details page, he is interested to see the different logs that are related to each task. Also, the user tends to remain on the tasks details page by browsing different tasks.

5.4.3 User Interface Transition Steps with Measured Probabilities

Dividing the user interface to states was straightforward. The sub-menu links on the main page form a natural and distinctive classification of the different states of MDS/SAS 5.0. Therefore, they are *Main*, *Network*, *Monitoring*, *Operational Configuration*, *Maintenance* and *User Maintenance* state, which are described in below:

Main state contains the selection of the further actions. Every time the user changes from one state to another, he has to go via this state.

Network state holds all operator's network infrastructure related things.

Monitoring state holds the reports of the activities inside MDS/SAS 5.0. For example, the incoming requests and the application statistics are observed in this state.

Operational Configuration state holds operations that are for handling the MDS/SAS 5.0 related parameters. For example, tracelevels and log file directories are set here. In addition, the service modules are operated in this state.

Maintenance state is for "maintaining" the system by exporting and importing the configurations. Also, the user can get different reports of the functions of MDS/SAS 5.0.

User Maintenance state is for handling the user access and profiles in MDS/SAS 5.0.

TFT provides a transition matrix describing the transitions from one state to another. Table 5.1 presents the matrix that was the output of executing this analysis with TFT. The matrix contains transitions from state A to state B and vice versa.

	Main	*Network*	*Monitoring*	*Oper.Conf.*	*Maintenance*	*UserMaint.*
Main	149	52	214	10	29	1
Network	48	219	4	0	0	0
Monitoring	218	0	982	0	7	0
Oper.Conf.	9	0	1	0	0	0
Maintenance	30	0	6	0	125	0
UserMaint.	1	0	0	0	0	0

Table 5.1: State transition matrix of MDS/SAS 5.0. The number of a particular state transition is marked in the matrix.

A state transition diagram with associated probabilites is a good representation for determining trends and patterns of using an application. By dividing the number of transitions from a state to a particular state by the total number of outgoing transitions, it is possible to calculate the probability of the transitions between states. The user action does not necessarily cause a state change, but it can be a transition to itself. Figure 5.2 is the transition diagram with associated probabilities of the matrix in Table 5.1.

It can be seen from Figure 5.2 that the state transitions violate the original definition of the states because there are transitions that are not going via the *Main* state. For example, the user changes state from *Monitoring* to *Maintenance* with a probability of 0,01. How should this be interpreted?

The answer is found from the browser's Back button, which makes it possible to follow the navigation path backwards without selecting any links. A state transition from *Monitoring* to *Maintenance* is actually going via *Main* state as can be seen in Figure 5.3. It is impossible to notice it automatically with the current analysis method because the Back button pressings are not recorded to the web server log. For that reason, we have to manually adjust the result by changing the state transition matrix. The changed matrix is in Table 5.2.

When we apply the adjusted state transition matrix to the state chart, the probabilities change a bit as can be seen in Figure 5.4.

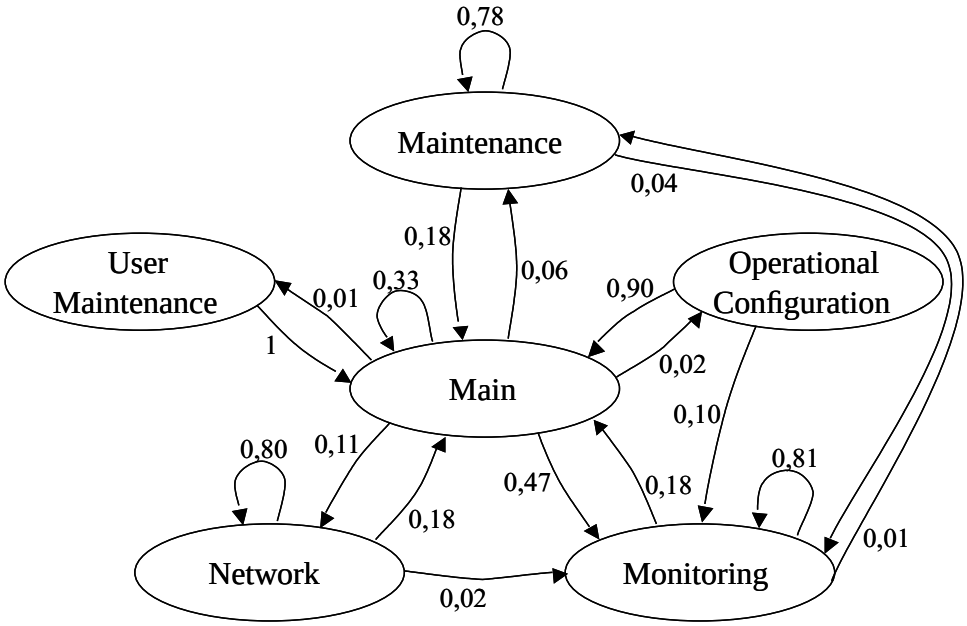

Figure 5.2: State chart of the usage of MDS/SAS 5.0. The application was divided to segments - *states* - according to the menu structure on the main page. The numbers in the chart are probabilities of the state transitions of the interface during the data collection period. Altogether, there were 2114 user actions that formed the sample of this analysis *(n=2114)*.

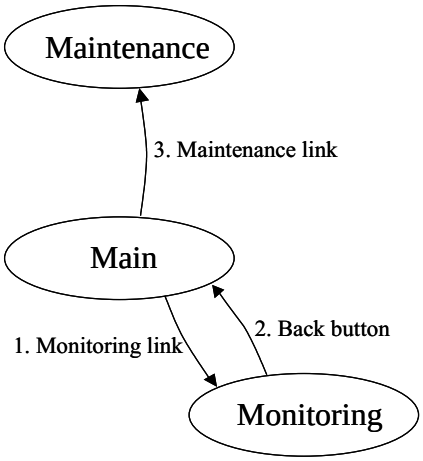

Figure 5.3: One possible state transition from Monitoring to Maintenance. The user action that changes the state is attached to the transition arrow with an order number.

	Main	*Network*	*Monitoring*	*Oper.Conf.*	*Maintenance*	*UserMaint.*
Main	149	52	**215**	10	**36**	1
Network	**52**	219	**0**	0	0	0
Monitoring	**225**	0	982	0	**0**	0
Oper.Conf.	**10**	0	**0**	0	0	0
Maintenance	**36**	0	**0**	0	125	0
UserMaint.	1	0	0	0	0	0

Table 5.2: Adjusted State transition matrix of MDS/SAS 5.0. The straight transitions between the states are changed to go via Main state as is the case in reality. The changed values are set in bold face.

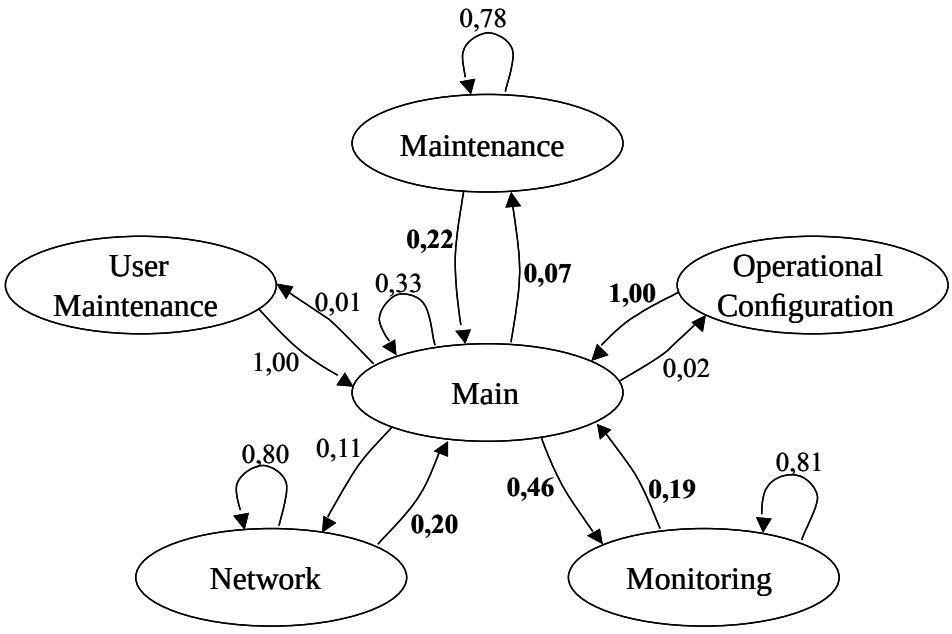

Figure 5.4: Adjusted state chart of the usage of MDS/SAS 5.0. This chart is based on the data in Table 5.2. The changed probabilities are set in bold face type.

The transition probabilities support the results of the previous analyses and provide some new information. The most probable transition from the *Main* state is the transition to *Monitoring* state. A somehow surprising result, which can not be seen from the previous analyses, is the probability of the transitions inside the *Main* state. It seems that the user is looking for some function by browsing the sub-menu links before entering a desired page. Moreover, those selections are not occurring in any specified order because they are not reported in User Action Sequence Frequencies analysis (see Section 5.4.2).

Even without knowing anything about the user, it seems to be possible to recognize his/her user group from the analysis results. The variety of the used tasks is rather small, which suggests that the person belongs to the second user group (see Section 4.2.2).

Chapter 6

Conclusions

This chapter begins with a summary of what was actually studied and performed in this thesis. The second section is a discussion part, where the results of this work are related to the bigger picture of the actual use logging. The final section is a description for the future research questions that came about during this thesis.

6.1 Summary

The actual use logging is a fruitful source to gather information about the user behaviour in the user interface. There is a clear practical need for developing tools for collecting and analysing the actual use logging events. However, because there is no formal methodology to consistently describe the graphical user interfaces, the collection has to use the methods that we have. In this study, a model and implementation for the collection and analysis was conducted by taking the HTML description of the application's user interface as a starting point.

After inquiring about the structure of the UI from the HTML description, the web server's access log was filtered against it. Consequently, only the intentional user actions were left. The semantically insignificant weblog was automatically transformed to contain particles, which could be enriched to useful semantic information. For this purpose, the study provided three different methods of analysis. They were selected to form a multipurpose toolbox that can be used in a variety of

situations.

In addition to the theoretical model, the study contains an implementation of it. The model was tested with a real life case. The implementation of the collection and analysis software was published under the GNU General Public License (GPL). The collection and analysis are separated in the implementation. Therefore, both, or either of them, can be included in some already existing software to enhance its features. Naturally, this has to follow the GPL policy.

6.2 Discussions

The model proved to be successful in providing interesting information with little human involvement. The results provided strong numeric information that help the case study company to analyse the usage of their software. For example, the main page of the software contained two links that were used very seldom and one link that was not used at all during the test period. As a result of all analyses, the next versions of the software's user interface could be even better suited for the user interests and different use tasks.

The idea of parsing the user interface structure from the HTML information was confirmed to be a good concept. It was possible to acquire a useful user interface structure, and apply it in conjunction with the weblog data. However, there were some limitations in the collectable elements. Because of the weblog collection, only the HTTP GET method requests were written into the log. Consequently, the HTML form information as well as button pressings were excluded from this study.

A distinguishing factor between the traditional hit-analysis software and the model presented in this thesis is the type of the collected and analysed elements. Here, the user activated elements are in focus instead of the downloaded user interface pages. For that reason, it is possible to execute analyses directly to the interface elements, and respectively get results that relate to them. All this concludes to a high level of automatisation in the collection and analysis while the results still provide highly semantical data.

The results from the User Interface Transition Steps with Measured Probabilities analysis suggest that it could be possible to recognize different user groups based on the transition probabilities. It would be interesting to compare the state charts of two users who belong to different user groups. If those state charts describe the characteristic use of the user groups well, they can be used as base reference charts in deciding to which group an unknown user belongs to. Unfortunately, there was no comparison material available during the project.

The sofware implementation, TFT, is technically bound to the browser-based interface. The log collection from a generic, e.g. Windows software, needs its own study, but the analyses could be the same than in this thesis. The collection of user interface events from the browser-based user interface was possible because the web server provided a method to collect the information between the user interface and the software core. It remains to be seen if the same is possible with Windows software.

There might be some interesting connections between the user interface state definition and its navigation map. Although the state definition was experimental in this thesis, it obviously was very close to the navigation map of the interface. Therefore, a designed navigation map helps to take TFT in use by defining the states for the User Interface Transition Steps with Measured Probabilities analysis beforehand.

6.3 Future Work

The research questions that arose during working with this thesis, can be divided into two groups. The first group consists of technical improvements and enhancements to TFT software. They are related to the browser-based user interfaces and can be seen as software development of TFT to improve the automatisation of the collection and analysis. The second group of the research questions is related to the actual use logging approach from greater perspective.

To improve the use of TFT, the user interface description collection should be automatised. In order to acquire the whole structure of the user interface, there should be a module with a breadth-first search algorithm to automatically collect

the description. The idea was shortly described in Section 3.2.2. Another project would be to include an automatic visualisation of the results to TFT. The User Interface Transition Steps with Measured Probabilities analysis would be rather straightforward to visualise with already existing Unix software tools.

If the actual use logging approach is wanted to integrate to the user interface design process from the very beginning, it needs some future work. Before we can suggest to use the navigation map of a user interface as a base in defining the states of the interface (see Section 6.2), more studies are needed. For example, different software can be studied in respect to their navigation maps and their states.

As was already mentioned in the previous section, it would be interesting to study if there is some mechanism in, e.g. Windows software, to automatically collect user events. Such research requires investigations of the techniques that are used in the window system. The research would be demanding but it would open the gate for actual use logging to today's "de facto" operating system.

I believe the results put forward in this work demonstrate that actual use logging approach can contribute to the field of usability. I showed how a generic web server log collection may be used to study the user behaviour in an application. The results were encouraging and analyses provided reliable data which was straightforward to present in visualised format. I hope this study encourages people involved in usability to see the potential of actual use logging; it is a method to better understand the user in the context of his/her application.

Bibliography

Badre, A. (1980), Chime: A knowledge-based computer-human interaction monitoring engine, Technical Report GIT-GVU-91-06, Georgia Institute of Technology, Atlanta, GA.

Balbo, S. (1996), Ema: Automatic analysis mechanism for the ergonomic evaluation of user interfaces, Technical report, Commonwealth Scientific and Industrial Research Organisation. CSIRO Technical report.

Buxton, W., Lamb, M., Sherman, D. & Smith, K. (1983), 'Towards a comprehensive user interface management system', *Computer Graphics* **17**(3).

Chen, C. (1993), 'Writing with collaborative hypertext: analysis and modelling', *Journal of the American Society for Information Science* **48**(11).

Comer, D. (1995), *Internetworking With TCP/IP, vol.1*, Prentice Hall International Editions, New Jersey, USA. ISBN 0-13-227836-7.

Cooley, R., Mobasher, B. & Srivastava, J. (1999), 'Data preparation for mining world wide web browsing patterns', *Knowledge and Information Systems* **1**.

Faraone, S. & Dorfman, D. (1987), 'Lag sequential analysis: Robust statistical methods', *Psychological Bulletin* **101**.

Finlay, J. & Wolf, K. (1995), User action graphing effort (usage), *in* 'Proceedings of HCI'95'.

Fisher, C. (1991), Protocol Analyst's Workbench: Design and evaluation of computer-aided protocol analysis, PhD thesis, Carnegie-Mellon University, Pittsburgh, PA.

Guzdial, M. (1996), Deriving software usage patterns from log files, Technical report, Commonwealth Scientific and Industrial Research Organisation. CSIRO Technical report.

Guzdial, M., Santos, P., Badre, A., Hudson, S. & Gray, M. (1994), 'Analyzing and visualizing log files: A computational science of usability'. *citeseer.nj.nec.com/guzdial94analyzing.html

Hilbert, D. & Redmiles, D. (1998), An approach to large-scale collection of application usage data over the internet, *in* 'Proceedings of the 20th International Conference on Software Engineering (ICSE'98).'.

Hilbert, D. & Redmiles, D. (2000), 'Extracting usability information from user interface events', *ACM Computing Surveys* **32**(4).

Hoppe, H. (1988), Task-oriented parsing: A diagnostic method to be used by adaptive systems, *in* 'Proceedings of CHI'88'.

Macleod, M. & Rengger, R. (1993), The development of drum: A software tool for video-assisted usability evaluation, *in* 'Proceedings of HCI'93'.

Net (1999), *Network Working Group, Request for Comments: 2616, Hypertext Transfer Protocol – HTTP/1.1*.

Nielsen, J. (1993), *Usability Engineering*, Academic Press, London, UK.

Olsen, D. & Redmiles, D. (1998), Interface usage measurements in a user interface management system, *in* 'Proceedings of UIST'88'.

Sanderson, P., Scott, J., Johnston, T., Mainzer, J., Watanabe, L. & James, J. (1994), 'Macshapa and the enterprise of exploratory sequential data analysis (esda)', *International Journal of Human-Computer Studies* **41**.

Sweeney, M., Maguire, M. & Shackel, B. (1993), 'Evaluating user-computer interaction: a framework', *International Journal of Man-Machine Studies* **38**.

W3C (1995), Logging control in w3c httpd. Document from the web page of World Wide Web Consortium. *http://www.w3.org/Daemon/User/Config/Logging.html

Weiler, P. (1993), Software for the usability lab: A sampling of current tools, *in* 'Proceedings of INTERCHI'93'.

Appendix A

Task Frequency Tool Source Code

WeblogHandler class:

```
/*************************************************************
 * Copyright (C) 2003 Pekka Partanen <pekka.partanen@iki.fi>  *
 * All rights reserved.                                       *
 *                                                            *
 * This program is free software; you can redistribute it     *
 * and/or modify it under the terms of the GNU General Public *
 * License. See the file COPYRIGHT for more information.      *
 *                                                            *
 * This program is distributed in the hope that it will be    *
 * useful, but WITHOUT ANY WARRANTY; without even the implied *
 * warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR    *
 * PURPOSE. See the GNU General Public License for more       *
 * details.                                                   *
 *                                                            *
 * This software is part of the Master's Thesis that was      *
 * written to the Department of Computer Science at Helsinki  *
 * University of Technology.                                  *
 *************************************************************/
package com.aul.tft;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.InputStreamReader;
import java.io.IOException;
import java.lang.Exception;

/**
 * <code>WeblogHandler</code> takes care of the weblog file. It
 * provides a method to get weblog lines one by one from the
 * file. This class reads all files from a specific weblog
 * directory.
 *
```

```
 * @author      <a href="pekka.partanen@iki.fi">Pekka Partanen</a>
 * @version     1.0
 * @see         BufferedReader
 * @see         File
 * @since       1.0
 */

public class WeblogHandler {

    /* weblog entries that were collected by web server
     */
    private static BufferedReader weblogFile = null;

    /* An array of filenames in the weblog directory
     */
    private static File[] fileNames;

    /* Counter of the filenames in the weblog directory
     */
    private static int fileNameCounter = 0;

    /**
     * Suppressed default constuctor
     */
    private WeblogHandler() {}

    /**
     * Constructor. Opens a <code>BufferedReader</code>
     * interface to a weblog file.
     *
     * @param filename the name of the weblog file
     * @see    BufferedReader
     * @since 1.0
     */
    public WeblogHandler(String dirName) throws Exception {

        try {
            LogWriter.logger.entering("WeblogHandler",
                                      "constructor");
            LogWriter.logger.fine("Opening weblogfile "+
                                  "directory :'" + dirName +"'");
            this.fileNames = (new File(dirName)).listFiles();

            if (this.fileNames == null) {
                String s = "directory '"+dirName+
                    "' does not contain files";
                LogWriter.logger.severe(s);
                throw new Exception(s);
            }

            weblogFile= new BufferedReader(
                        new InputStreamReader(
                         new FileInputStream(this.fileNames[
                                        this.fileNameCounter])));
            ++this.fileNameCounter;
```

```
            LogWriter.logger.fine("Opened weblogfile :'"+
          (this.fileNames[this.fileNameCounter - 1]).toString()+
                                        "'");

        } catch (Exception e) {
            LogWriter.logger.severe(e.getMessage());
            throw e;
        }
    }


    /**
     * Gets a single line from the weblog file. Returns an empty
     * <code>String</code> if the weblog file is empty.
     *
     * @return weblog line
     * @see    BufferedReader
     * @since  1.0
     */
    public String getLine() throws Exception {
        try {
            LogWriter.logger.entering("WeblogHandler",
                                      "getLine");
            String line;
            String value;

            // If the file is read, a new file is used. But if
            // the read file was the last one then an empty
            // string is returned.
            if ((line = weblogFile.readLine()) == null) {
                if (this.fileNameCounter >=
                    this.fileNames.length) {
                    LogWriter.logger.fine("No more weblogfiles"+
                                          " to read");
                    value = "";
                } else {
                    weblogFile = new BufferedReader(
                                     new InputStreamReader(
                                 new FileInputStream(this.fileNames[
                                         this.fileNameCounter])));
                    ++this.fileNameCounter;
                }

                if ((line = weblogFile.readLine()) == null) {
                    LogWriter.logger.fine("No more weblogfiles"+
                                          " to read");
                    value = "";
                } else
                    value = line;
            } else
                value = line;

            LogWriter.logger.exiting("WeblogHandler",
                                     "getLine");
```

```
            return value;

        } catch (IOException e) {
            LogWriter.logger.severe("Catched␣IOException␣:'" +
                                    e.getMessage() + "'");
            throw e;
        } catch (Exception e) {
            LogWriter.logger.severe("Catched␣Exception␣:'" +
                                    e.getMessage() + "'");
            throw e;
        }
    }
}
```

UIDescrHandler class:

```
/*************************************************************
 * Copyright (C) 2003 Pekka Partanen <pekka.partanen@iki.fi>     *
 * All rights reserved.                                          *
 *                                                               *
 * This program is free software; you can redistribute it        *
 * and/or modify it under the terms of the GNU General Public    *
 * License. See the file COPYRIGHT for more information.         *
 *************************************************************/
package com.aul.tft;

import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.InputStreamReader;
import java.util.regex.Pattern;
import java.util.regex.Matcher;
import java.util.BitSet;
import java.util.Hashtable;
import java.util.Vector;

/**
 * UIDescrHandler takes care of the user interface description.
 * It reads the format of the description file and provides a
 * set of methods for other classes to access the description
 * file. The file is read to a <code>Hashtable</code> for faster
 * reference.
 *
 * @author      <a href="pekka.partanen@iki.fi">Pekka Partanen</a>
 * @version     1.0
 * @see         BitSet
 * @see         BufferedReader
 * @see         Hashtable
 * @see         Matcher
 * @see         Pattern
 * @see         Vector
 * @since       1.4
 */

public class UIDescrHandler {

    private static Hashtable keyTable = new Hashtable();
    private static Hashtable valueTable = new Hashtable();
    private static Hashtable keyAndPageMappingTable =
        new Hashtable();
    private static Hashtable keyToParentLinkName =
        new Hashtable();
    private static Hashtable linkNameToKey = new Hashtable();
    private static String pageNameSeparator;
    private static String linkNameSeparator;

    // Suppressed default constructor
    private UIDescrHandler() {}

    /**
```

```
     * Constructor. The description file name is passed as a
     * parameter and the file is read to a
     * <code>Hashtable</code>.
     *
     * @param filename the name of the user interface
     *                 description file.
     * @since 1.0
     */
    public UIDescrHandler(String filename) throws Exception{

        try {
            LogWriter.logger.entering("UIDescrHandler",
                                      "constructor");

            int bitIndex = 0;
            BufferedReader uiFile;
            Matcher m, n;
            Pattern p, q;
            String key;
            String line;
            String parentLink = "";

            // Storing page and link name separator characters
            // to member variables
            this.pageNameSeparator =
              ConfReader.getValue(Parameters.PAGENAMESEPARATOR);
            this.linkNameSeparator =
              ConfReader.getValue(Parameters.LINKNAMESEPARATOR);

            uiFile = new BufferedReader(
                       new InputStreamReader(
                         new FileInputStream(filename)));

            LogWriter.logger.fine("Opened :'" + filename + "'");

            // The parsing regexp is read from the configuration
            // file. Each matching line of the description file
            // is parsed and stored to Hashtable.
            p = Pattern.compile(ConfReader.getValue(
                                  Parameters.UIDSCRPTREGEXP));
            q = Pattern.compile("([a-zA-Z0-9_ ]*) [->].*");
            while ((key = uiFile.readLine()) != null) {
                m = p.matcher(key);
                n = q.matcher(key);

                if (n.matches()) {
                    parentLink = (n.group(1)).trim();
                    LogWriter.logger.fine("Matched q. Line :'"+
                                          key+"'");
                    LogWriter.logger.fine("Page name :'"+
                                          parentLink+"'");
                } else if (m.matches()) {
                    LogWriter.logger.fine("Matched m. Line :'"+
                                          key+"'");
                    line = key;
```

```
                    key = m.group(Parameters.SELECTED_GROUP);
                    // Dynamic content is abstracted
                    key =
                      CommonTools.convertRequestlineToName(key);
                    // The page name and the link text are
                    // stored to hashtable
                    storePageAndLink(key, line);

                    // Store the key to hashtable with name of
                    // the parent link
                    keyToParentLinkName.put((String)key,
                                            (String)parentLink);

                    // Key values are also inserted to
                    // Hashtable. If the key value is already in
                    // the table then we don't want to store it
                    // again.
                    if (!isKeyUIElement(key)) {
                        storeKey(key, bitIndex);
                        ++bitIndex;
                    }
                } else {
                    LogWriter.logger.fine("Does not match.Line"+
                                          " :'"+key+"'");
                }
            }

            LogWriter.logger.exiting("UIDescrHandler",
                                     "constructor");
        } catch (Exception e) {
            LogWriter.logger.severe(e.getMessage());
            throw e;
        }
    }


    /**
     * Stores the key and bit index to two
     * <code>Hashtable</code>s. One is keyed with a
     * bitIndex-variable while the other with a key-variable.
     *
     * @param key      the abstracted user interface description
     *                 line
     * @param bitIndex the number of the user interface element
     * @see    BitSet
     * @see    Hashtable
     * @since 1.0
     */
    private static void storeKey(String key, int bitIndex){

        try {
            LogWriter.logger.entering("UIDescrHandler",
                                      "storeKey");

            BitSet bitset = new BitSet();
```

```
            // bitIndex is set to BitSet instance
            bitset.set(bitIndex);
            keyTable.put(key, bitset);
            valueTable.put(bitset, key);

            LogWriter.logger.exiting("UIDescrHandler",
                                     "storeKey");
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }


    /**
     * Parses the page name and link text from the user
     * interface description line. Stores values to a
     * <code>Vector</code> and inserts it to a
     * <code>Hashtable</code> with a previously with key.
     *
     * @param key  the key for Hashtable
     * @param line the user interface description line that
     *             contains the link text and page name
     * @see    Vector
     * @see    Hashtable
     */
    private static void storePageAndLink(String key,
                                         String line)
        throws Exception {

        try {
            LogWriter.logger.entering("UIDescrHandler",
                                      "storePageAndLink");
            String pageName = "";
            String linkText = "";
            Vector pageAndLink =
                new Vector(Parameters.PAGE_AND_LINK_ELEMS);

            // Page name and link text are parsed from the user
            // interface description file
            pageName =
               line.substring(line.indexOf(pageNameSeparator)+1,
                              line.lastIndexOf(pageNameSeparator));

            linkText =
              line.substring(line.indexOf(linkNameSeparator)+1,
                              line.lastIndexOf(linkNameSeparator));

            LogWriter.logger.fine("page name :'"+pageName+
                                  "' and link text :'"+
                                  linkText+"'");

            // Page name and link text are stored to a vector,
            // which is inserted to a Hashtable
            pageAndLink.add(Parameters.PAGE_ENTRY, pageName);
```

```
            pageAndLink.add(Parameters.LINK_ENTRY, linkText);
            keyAndPageMappingTable.put(key, pageAndLink);

            // Variable key is stored as a value to
            // hashtable. Variable linkText is used as a key.
            linkNameToKey.put(linkText, key);

            LogWriter.logger.exiting("UIDescrHandler",
                                     "storePageAndLink");
        } catch (Exception e) {
            LogWriter.logger.severe(e.getMessage());
            throw e;
        }
    }


    /**
     * Returns the boolean value depending if the key is in the
     * <code>Hashtable</code>.
     *
     * @param  key the key value of a Hashtable entry
     * @return <code>true</code> if the key was in the Hashtable
     * @see    Hashtable
     * @since  1.0
     */
    public static boolean isKeyUIElement(String key) {
        return keyTable.containsKey(key);
    }


    /**
     * Returns the <code>BitSet</code> value of the class
     * variable keyTable.
     *
     * @param  key the key value of a Hashtable entry
     * @return BitSet value of the Hashtable keyTable
     * @see    BitSet
     * @see    Hashtable
     * @since  1.0
     */
    public static BitSet getKeyValue(String key) {
        return (BitSet)keyTable.get(key);
    }


    /**
     * Gets the converted name of the request line that is
     * associated to a <code>BitSet</code> value.
     *
     * @param  key the key value of a Hashtable entry
     * @return <code>true</code> if the key was in the Hashtable
     * @see    BitSet
     * @see    Hashtable
     * @since  1.0
     */
```

```
public static String getKeyName(BitSet value) {
    return (String)valueTable.get(value);
}


/**
 * Gets the name of the key, which is mapped to a link text.
 *
 * @see    Hashtable
 * @since 1.0
 */
public static String
    getKeyToLinkTextMapping(String linkText) {
    return (String)linkNameToKey.get(linkText);
}


/**
 * Gets a vector containing the name of the page and link to
 * where the converted http request line is associated to.
 *
 * @param key converted http request line
 * @see    Vector
 * @since 1.0
 */
public static Vector getKeytoPageMapping(String key) {
    return (Vector)keyAndPageMappingTable.get(key);
}


/**
 * Gets the keyAndPageMappingTable.
 *
 * @param key converted http request line
 * @see    Vector
 * @since 1.0
 */
public static Hashtable getKeytoPageMappingTable() {
    return keyAndPageMappingTable;
}


/**
 * Gets a string containing the name of the parent link that
 * points to the page in where <code>key</code> exists.
 *
 * @param key converted http request line
 * @see    Hashtable
 * @since 1.0
 */
public static String
    getKeyToParentLinkNameMapping(String key) {
    return (String)keyToParentLinkName.get(key);
}
```

```
    /**
     * Gets keyToParentLinkName hashtable.
     *
     * @see     Hashtable
     * @since 1.0
     */
    public static Hashtable getKeyToParentLinkNameHashtable() {
        return keyToParentLinkName;
    }
}
```

UIAndWeblogMatcher class:

```
/*************************************************************
 * Copyright (C) 2003 Pekka Partanen <pekka.partanen@iki.fi>  *
 * All rights reserved.                                       *
 *                                                            *
 * This program is free software; you can redistribute it     *
 * and/or modify it under the terms of the GNU General Public *
 * License. See the file COPYRIGHT for more information.      *
 *************************************************************/
package com.aul.tft;

import java.io.File;
import java.io.FileOutputStream;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

/**
 * Matches the user interface description to the weblog entries.
 * In the other words, this class filters the intentional user
 * actions from the weblog noise. The filtered actions are
 * stored to a file in a key format, which can be used in
 * analysis.
 *
 * @author      <a href="pekka.partanen@iki.fi">Pekka Partanen</a>
 * @version     1.0
 * @see         File
 * @see         Matcher
 * @see         Pattern
 * @see         PrintWriter
 * @since       1.4
 */

public class UIAndWeblogMatcher {

    /* Filtered HTTP query lines containing only the intentional
     * user actions
     */
    private PrintWriter filteredEntries;

    /**
     * Suppressed default constuctor
     */
    private UIAndWeblogMatcher() {}

    /**
     * Constructor. Opens a file for storing the filtered user
     * actions.
     *
     * @param uiElemFile the user interface description file
     * @see    PrintWriter
     * @since 1.0
     */
```

```
    public UIAndWeblogMatcher(File uiElemFile)
        throws Exception {
        try {
            LogWriter.logger.entering("UIAndWeblogMatcher",
                                      "constructor");

            filteredEntries = new PrintWriter(
                              new OutputStreamWriter(
                    new FileOutputStream(uiElemFile, true)));

            LogWriter.logger.exiting("UIAndWeblogMatcher",
                                      "constructor");
        } catch (Exception e) {
            LogWriter.logger.severe(
              "Catched an exception in UIAndWeblogMatcher"+
              "constructor");
            throw e;
        }
    }

    /**
     * Matches the user interface description file and weblog
     * entries. If matches are found, they are written to a file
     * in key format. In that format, the file entries can be
     * used in UIDescrHandler class.
     *
     * @param weblogHandler reference to the weblog handling
     *                      class instance
     * @see UIDescrHandler
     * @since 1.4
     */
    public void matchUIAndWeblog(WeblogHandler weblogHandler)
        throws Exception {

        try {
            LogWriter.logger.entering("UIAndWeblogMatcher",
                                      "matchUIAndWeblog");
            String keyCand = "";
            Matcher m;
            Pattern p;

            p = Pattern.compile(ConfReader.getValue(
                                Parameters.WEBLOGREGEXP));
            LogWriter.logger.fine("Compiled regexp :'" +
                        p.pattern()+"' for parsing the weblog");

            // Each line of the weblog is parsed and converted
            // to the same representation than the user
            // interface description file lines. Parsed request
            // lines are compared to converted user interface
            // description lines to check if the request line is
            // one of the user interface elements
            while ((keyCand = weblogHandler.getLine()) != "") {
                LogWriter.logger.fine("keyCand :'"+keyCand+"'");
                m = p.matcher(keyCand);
```

```
                if (m.matches()) {
                    keyCand = m.group(Parameters.SELECTED_GROUP);
                    LogWriter.logger.fine("Parsed␣key␣"+
                                          "candidate␣:'" +
                                          keyCand + "'");
                    // Parameter values are abstracted
                    keyCand =
                        CommonTools.convertRequestlineToName(
                                                    keyCand);
                    LogWriter.logger.finest(
                      "Key␣candidate␣after␣abstraction␣:'"+
                      keyCand+"'");

                    // If weblog entry is a user interface
                    // element, it is written to a file for
                    // later use
                    if (UIDescrHandler.isKeyUIElement(keyCand)) {
                        LogWriter.logger.fine("Key␣'" + keyCand+
                                        "'␣matched␣to␣UI␣element");
                        filteredEntries.println(keyCand);

                    }
                }
            }

            LogWriter.logger.exiting("UIAndWeblogMatcher",
                                     "constructor");
        } catch (Exception e) {
            LogWriter.logger.severe(
              "Catched␣an␣exception␣in␣matchUIAndWeblog");
            // Closing of the filtered user actions file
            filteredEntries.close();
            throw e;
        }

        filteredEntries.close();
    }
}
```

TransitionStepAnalysis class:

```
/*************************************************************
 * Copyright (C) 2003 Pekka Partanen <pekka.partanen@iki.fi>  *
 * All rights reserved.                                       *
 *                                                            *
 * This program is free software; you can redistribute it     *
 * and/or modify it under the terms of the GNU General Public *
 * License. See the file COPYRIGHT for more information.      *
 *************************************************************/
package com.aul.tft;

import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.InputStreamReader;
import java.lang.Integer;
import java.util.Enumeration;
import java.util.Hashtable;
import java.util.Vector;

/**
 * Class for transition step analysis. Transitions between the
 * application states are counted. Here, the states are defined
 * as the links on the main page.
 *
 * @author      <a href="pekka.partanen@iki.fi">Pekka Partanen</a>
 * @version     1.0
 * @see         BufferedReader
 * @see         Enumeration
 * @see         Hashtable
 * @see         Vector
 * @since       1.0
 */

public class TransitionStepAnalysis {

    /* Filtered user actions
     */
    private BufferedReader uiFile;

    /* Name of the root page
     */
    private String rootPage;

    /* A hashtable in where an abstracted HTTP query line is a
     * key and the page name is the value
     */
    private Hashtable keyAndPageMappingTable;

    /* Transitions between the states. Vector from state to
     * state is a key and the number of its occurrences is the
     * value
     */
    private Hashtable transitions = new Hashtable();
```

```
/**
 * Suppressed default constructor
 */
private TransitionStepAnalysis() {}


/**
 * Constructor. The root page name and the intentional user
 * action file is read.
 *
 * @param userActions the file where the intentional user
 *                    action keys are stored
 * @see    BufferedReader
 * @see    Hashtable
 * @since 1.0
 */
public TransitionStepAnalysis(String userActions)
    throws Exception {

    try {
        LogWriter.logger.entering("TransitionStepAnalysis",
                                  "constructor");
        uiFile = new BufferedReader(
                 new InputStreamReader(
                  new FileInputStream(userActions)));

        rootPage = ConfReader.getValue(Parameters.ROOTPAGE);
        keyAndPageMappingTable =
            UIDescrHandler.getKeytoPageMappingTable();

        LogWriter.logger.exiting("TransitionStepAnalysis",
                                 "constructor");
    } catch (Exception e) {
        LogWriter.logger.severe("Catched exception :'"+
                                e.getMessage()+"'");
        throw e;
    }
}


/**
 * This method runs the transition step analysis. At first,
 * two adjacent intentional user actions are read from file.
 * Their parent pages – states – are searched and are stored
 * to <code>Hashtable</code> in <code>storeTransition</code>
 * method. After the user action file is read to the end,
 * the transition states are printed with their occurrence
 * information.
 *
 * @see    Enumeration
 * @see    Hashtable
 * @see    Vector
 * @since 1.0
 */
public void analyse() throws Exception {
```

```
        try {
            LogWriter.logger.entering("TransitionStepAnalysis",
                                      "analyse");
            String key;
            String pageA = "";
            String pageB = "";

            // All user actions are read and their native parent
            // pages are searched. Then transitions from native
            // page x to native page y are stored to a hashtable

            if ((key = uiFile.readLine()) == null)
                throw new Exception("Use action file is empty");

            pageA = searchParents(key);

            while ((key = uiFile.readLine()) != null) {
                pageB = searchParents(key);

                storeTransition(pageA, pageB);
                pageA = pageB;
            }

            Vector temp =
                new Vector(Parameters.STATE_AND_OCCURRENCES);
            for (Enumeration e =
                  this.transitions.keys(); e.hasMoreElements();) {
                temp = (Vector)e.nextElement();
                System.out.println(temp+" :'"+
                    (Integer)this.transitions.get((Vector)temp)+
                                    "'");
            }

            LogWriter.logger.exiting("TransitionStepAnalysis",
                                     "analyse");
        } catch (Exception e) {
            LogWriter.logger.severe("Catched an exception in "+
                                    "TransitionStepAnalysis");
            throw e;
        }
    }


    /**
     * Searches all parent links that the parameter
     * <code>key</code> has. The search is done until the "root
     * page" is found. At first, the key is searched from
     * UIDescrHandler's hashtable keyAndPageMappingTable, which
     * gives the name of the link that took the user to the page
     * where this link was pressed. Then, the name of the link
     * is searched from UIDescrHandler's keyAndPageMappingTable
     * hashtable and when it is found, the appropriate key is
     * read and it's parent link is searched. This continues
     * until the parent link name is the "root page".
```

```
     *
     * @param  key HTTP query line in abstracted format
     * @return a state to which <code>key</code> belongs to
     * @since  1.0
     */
    private String searchParents(String key) throws Exception {

        try {
            LogWriter.logger.entering("searchParents",
                                      "constructor");

            String parentLink =
               UIDescrHandler.getKeyToParentLinkNameMapping(key);
            String prevLink = "";
            String parentKey;

            while (!parentLink.equals(this.rootPage)) {

                prevLink = parentLink;
                // Here we search the corresponding key value to
                // the parent link text
                parentKey =
              UIDescrHandler.getKeyToLinkTextMapping(parentLink);
                parentLink =
         UIDescrHandler.getKeyToParentLinkNameMapping(parentKey);
            }


            LogWriter.logger.exiting("TransitionStepAnalyse",
                                     "searchParents");

            if (!prevLink.equals(""))
                return prevLink;
            else
                return parentLink;

        } catch (Exception e) {
            LogWriter.logger.severe("Catched an exception in "+
                                    "searchParents :'" +
                                    e.getMessage()+"'");
            throw e;
        }
    }

    /**
     * Stores the transition from state A to state B to a
     * hashtable. If a certain transition already exists in the
     * hashtable, the value is initialised to '1'. Otherwise,
     * the value is increased by '1'.
     *
     * @param stateA  The source state
     * @param stateB  The destination state
     * @see   Hashtable
     * @see   Vector
     * @since 1.0
```

```
     */
    private void storeTransition(String stateA, String stateB) {

        try {
            LogWriter.logger.entering("storeTransition",
                                      "constructor");

            Vector pagesAandB = new Vector(2);
            Integer value;

            LogWriter.logger.fine("Storing values stateA :'"+
                                  stateA+"' stateB :'"+stateB+"'");
            pagesAandB.add(Parameters.STATE_POSITION,
                           (String)stateA);
            pagesAandB.add(Parameters.OCCURRENCE_POSITION,
                           (String)stateB);

            // If the key already exists in the hashtable, the
            // value is increased by one. Otherwise, the value is
            // initialised to '1'
          if (this.transitions.containsKey((Vector)pagesAandB)) {
              LogWriter.logger.fine("key existed");
              value =
             (Integer)this.transitions.get((Vector)pagesAandB);
              value = new Integer(value.intValue() + 1);
            } else {
              LogWriter.logger.fine("key does not exist");
              value = new Integer(1);
            }

            LogWriter.logger.fine("Storing value :'"+
                                  value.toString()+"'");
            this.transitions.put((Vector)pagesAandB.clone(),
                                 (Integer)value);

            LogWriter.logger.exiting("TransitionStepAnalyse",
                                     "storeTransition");
        } catch (Exception e) {
            LogWriter.logger.severe("Catched an exception in "+
                                    "storeTransition :'"+
                                    e.getMessage()+"'");
        }
    }
}
```

Runner class:

```
/*************************************************************
 * Copyright (C) 2003 Pekka Partanen <pekka.partanen@iki.fi>  *
 * All rights reserved.                                       *
 *                                                            *
 * This program is free software; you can redistribute it     *
 * and/or modify it under the terms of the GNU General Public *
 * License. See the file COPYRIGHT for more information.      *
 *************************************************************/
package com.aul.tft;

import java.io.File;
import java.util.Hashtable;

/**
 * Executor of the collection and analysis classes.
 *
 * @author      <a href="pekka.partanen@iki.fi">Pekka Partanen</a>
 * @version     1.0
 * @see         File
 * @see         Hashtable
 * @since       1.0
 */

public final class Runner {

    /**
     * Suppressed default constructor
     */
    private Runner() {}

    /** Main program
     * @param args command line arguments (not used)
     */
    public static void main(String args[]) {

        try {
            ConfReader confReader = new ConfReader();
            LogWriter logWriter = new LogWriter();
            logWriter.setLogLevel(ConfReader.getValue(
                                        Parameters.DBGLEVEL));
            UIDescrHandler uiDescrHandler =
                new UIDescrHandler(ConfReader.getValue(
                                        Parameters.UIDSCRPT));
            File uiElemFile =
                new File(ConfReader.getValue(
                                    Parameters.UIELEMFILE));

            // the user action file is reused if it exists
            if (!uiElemFile.exists()) {
                logWriter.logger.fine("Parsed UI element does"+
                                      "not exists");
                UIAndWeblogMatcher matcher =
                    new UIAndWeblogMatcher(uiElemFile);
```

```
                WeblogHandler weblogHandler =
                    new WeblogHandler(ConfReader.getValue(
                                            Parameters.WEBLOGDIR));
                matcher.matchUIAndWeblog(weblogHandler);
            } else
                logWriter.logger.fine("Parsed␣UI␣element␣"+
                                      "already␣exists");

            NumberOfHitsAnalysis nohAnalysis =
                new NumberOfHitsAnalysis();

            Hashtable freqs =
                nohAnalysis.runAnalysis(Integer.parseInt(
                        ConfReader.getValue(Parameters.SEQLGTH)));

            nohAnalysis.orderKeysbyFreqstoArray(freqs);

            // Transition step analysis
            TransitionStepAnalysis transitionSteps =
                new TransitionStepAnalysis(ConfReader.getValue(
                                        Parameters.UIELEMFILE));

            transitionSteps.analyse();

            LogWriter.logger.exiting("Runner", "main");
        } catch (Exception e) {
            LogWriter.logger.severe(e.getMessage());
            System.out.println(e.getMessage());
        }
    }
}
```

Parameters class:

```
/*************************************************************
 * Copyright (C) 2003 Pekka Partanen <pekka.partanen@iki.fi>     *
 * All rights reserved.                                          *
 *                                                               *
 * This program is free software; you can redistribute it        *
 * and/or modify it under the terms of the GNU General Public    *
 * License. See the file COPYRIGHT for more information.         *
 *************************************************************/
package com.aul.tft;

/**
 * This class contains the constant values. For example, the
 * configuration file's parameter names are stored here.
 *
 * @author      <a href="pekka.partanen@iki.fi">Pekka Partanen</a>
 * @version     1.0
 * @since       1.0
 */

public class Parameters{

    // ConfReader values
    static final String CONFFILE = "tft.properties";
    static final String TFTLOGFILE = "tftlogfile";
    static final String WEBLOGDIR = "weblogdir";
    static final String WEBLOGREGEXP = "weblogregexp";
    static final String DBGLEVEL = "dbglevel";
    static final String SEQLGTH = "seqlgth";
    static final String UIDSCRPT = "uidscrpt";
    static final String UIDSCRPTREGEXP = "uidscrptregexp";
    static final String UIELEMFILE = "uielemfile";
    static final String PAGENAMESEPARATOR = "pagenameseparator";
    static final String LINKNAMESEPARATOR = "linknameseparator";
    static final String ROOTPAGE = "rootpage";

    // Configuration file's uidscrptparsingregexp parameter for
    // the group that is declared with parenthesis
    static final int SELECTED_GROUP = 1;

    // Length of the vector that stores the page name and link
    // text and their vector positions
    static final int PAGE_AND_LINK_ELEMS = 2;
    static final int PAGE_ENTRY = 0;
    static final int LINK_ENTRY = 1;

    // Used in LogWriter class for opening the log file so that
    // the new log messages are appended to the file
    static final boolean APPEND = true;

    // Used in TransitionStepAnalysis for initialising a vector
    // to where the state and its number of occurrences is
    // stored
    static final int STATE_AND_OCCURRENCES = 2;
```

```
    static final int STATE_POSITION = 0;
    static final int OCCURRENCE_POSITION = 1;

    /**
     * Suppressed default constructor
     */
    private Parameters() {}
}
```

LogWriter class:

```
/*************************************************************
 * Copyright (C) 2003 Pekka Partanen <pekka.partanen@iki.fi>     *
 * All rights reserved.                                          *
 *                                                               *
 * This program is free software; you can redistribute it        *
 * and/or modify it under the terms of the GNU General Public    *
 * License. See the file COPYRIGHT for more information.         *
 *************************************************************/
package com.aul.tft;

import java.util.logging.Logger;
import java.util.logging.FileHandler;
import java.util.logging.Level;
import java.util.logging.SimpleFormatter;

/**
 * LogWriter is a wrapper for <code>Logger</code> class. It
 * provides a static variable <code>logger</code> for other
 * methods for log message writing.
 *
 * @author      <a href="pekka.partanen@iki.fi">Pekka Partanen</a>
 * @version     1.0
 * @see         FileHandler
 * @see         Logger
 * @see         SimpleFormatter
 * @since       1.4
 */

public class LogWriter {

    /* Reference for <code>Logger</code> class
     */
    public static Logger logger;

    /* File for log writing
     */
    private static FileHandler fh;

    /* Formatter for the <code>Logger</code> method
     */
    private static SimpleFormatter sf;

    /**
     * Default constructor for logwriter.
     *
     * @see    Logger
     * @since 1.4
     */
    public LogWriter () {
        try {
            sf = new SimpleFormatter();
            logger = Logger.getLogger("tftsystem");
            fh = new FileHandler(ConfReader.getValue(
```

```
                                                Parameters.TFTLOGFILE),
                                        Parameters.APPEND);
                fh.setFormatter(sf);
                logger.addHandler(fh);
            } catch(Exception e) {
                System.err.println(e.getMessage());
            }
        }


        /**
         * Sets the log level. Maps the following values to
         * corresponding level values:
         * <ul>
         * <li>SEVERE
         * <li>WARNING
         * <li>INFO
         * <li>CONFIG
         * <li>FINE
         * <li>FINEST
         * </ul>
         * <p>
         * If none of the values above match the parameter's value,
         * <code>ALL</code> value is used for log level.
         *
         * @param level value from the configuration file for the
         *              log level
         * @since 1.4
         */
        public void setLogLevel(String level) {

            if (level.equals("SEVERE"))
                logger.setLevel(Level.SEVERE);
            else if (level.equals("WARNING"))
                logger.setLevel(Level.WARNING);
            else if (level.equals("INFO"))
                logger.setLevel(Level.INFO);
            else if (level.equals("CONFIG"))
                logger.setLevel(Level.CONFIG);
            else if (level.equals("FINE"))
                logger.setLevel(Level.FINE);
            else if (level.equals("FINER"))
                logger.setLevel(Level.FINER);
            else if (level.equals("FINEST"))
                logger.setLevel(Level.FINEST);
            else
                logger.setLevel(Level.ALL);

            logger.config("Logger␣initialized␣to␣level␣:'"+
                          logger.getLevel()+"'");
        }
    }
```

FreqBitSet class:

```
/*************************************************************
 * Copyright (C) 2003 Pekka Partanen <pekka.partanen@iki.fi>  *
 * All rights reserved.                                       *
 *                                                            *
 * This program is free software; you can redistribute it     *
 * and/or modify it under the terms of the GNU General Public *
 * License. See the file COPYRIGHT for more information.      *
 *************************************************************/
package com.aul.tft;

import java.util.Vector;
import java.util.BitSet;

/**
 * A data structure for storing information of the user action
 * sequence and a number of their occurrences. This class
 * implements <code>Comparable</code> class because the
 * occurrences are wanted to be output in order.
 *
 * @author      <a href="pekka.partanen@iki.fi">Pekka Partanen</a>
 * @version     1.0
 * @see         Comparable
 * @see         Integer
 * @see         Vector
 * @since       1.2
 */

public class FreqBitSet implements Comparable {

    /* Variable for storing the number of keyVector occurrences
     */
    private Integer freq;

    /* Vector for storing the user action sequence
     */
    private Vector keyVector;

    /**
     * Suppressed default constructor
     */
    private FreqBitSet() {}

    /**
     * Constructor that stores the parameters to class variables.
     *
     * @param freq      number of occurrences of keyVector
     * @param keyVector user action sequence
     * @since 1.0
     */
    public FreqBitSet(Integer freq, Vector keyVector) {

        LogWriter.logger.entering("FreqBitSet", "constructor");
        LogWriter.logger.finest("Received parameters freq :'"+
```

```
                                freq+"' and keyVector :'"+
                                keyVector+"'");

        if (freq == null | keyVector == null) {
            NullPointerException npe =
                new NullPointerException();
            LogWriter.logger.throwing("FreqBitSet",
                                      "constructor", npe);
            throw npe;
        }

        this.freq = freq;
        this.keyVector = (Vector)keyVector.clone();
        LogWriter.logger.exiting("FreqBitSet", "constructor");
    }


    /**
     * Gets the class variable <code>freq</code>.
     *
     * @return number of the occurrences
     * @since  1.0
     */
    public Integer freq() {
        LogWriter.logger.entering("FreqBitSet", "freq");
        LogWriter.logger.exiting("FreqBitSet", "freq");
        return freq;
    }


    /**
     * Gets the class variable <code>keyVector</code>.
     *
     * @return <code>Vector</code> that holds the user action
     * sequence
     * @since  1.0
     */
    public Vector keyVector () {
        LogWriter.logger.entering("FreqBitSet", "keyVector");
        LogWriter.logger.exiting("FreqBitSet", "keyVector");
        return keyVector;
    }


    /**
     * Gets the size of the <code>keyVector</code>.
     *
     * @return size of the <code>keyVector</code>
     * @since  1.0
     */
    public int keyVectorSize() {
        LogWriter.logger.entering("FreqBitSet", "keyVectorSize");
        LogWriter.logger.exiting("FreqBitSet", "keyVectorSize");
        return keyVector.size();
    }
```

```
/**
 * Implemented new <code>equals</code> method.
 *
 * @param  comparable <code>Object</code>
 * @return <code>true</code> if the <code>Object</code>
 * equals to this class
 * @since  1.0
 */
public boolean equals(Object o) {
    LogWriter.logger.entering("FreqBitSet", "equals");
    if (! (o instanceof FreqBitSet)) {
        LogWriter.logger.exiting("FreqBitSet", "equals");
        return false;
    }

    FreqBitSet fbs = (FreqBitSet)o;
    LogWriter.logger.exiting("FreqBitSet", "equals");
    return fbs.freq.equals(freq);
}


/**
 * Implemented new <code>hashCode</code> method.
 *
 * @return new hashcode
 * @since 1.0
 */
public int hashCode() {
    LogWriter.logger.entering("FreqBitSet", "hashCode");
    LogWriter.logger.exiting("FreqBitSet", "hashCode");
    return 16*freq.hashCode();
}


/**
 * Implemented new <code>toString</code> method.
 *
 * @return <code>String</code> presentation of this class
 * @since  1.0
 */
public String toString() {
    LogWriter.logger.entering("FreqBitSet", "toString");
    String temp = new String();
    temp = freq + "␣";
    for (int i = 0; i < keyVector.size(); ++i)
        temp = temp + "␣" + (BitSet)keyVector.get(i);
    LogWriter.logger.finest("Returning␣value␣:'"+temp+"'");
    LogWriter.logger.exiting("FreqBitSet", "toString");
    return temp;
}


/**
```

```
     * Comparing <code>Object</code> to this class.
     *
     * @param  <code>Object</code> to compare to this class
     * @return the value 0 if the argument string is equal to
     * this string; a value less than 0 if this string is
     * lexicographically less than the string argument; and a
     * value greater than 0 if this string is lexicographically
     * greater than the string argument.
     * @since  1.0
     */
    public int compareTo(Object o) {

        LogWriter.logger.entering("FreqBitSet", "compareTo");
        FreqBitSet fbs = (FreqBitSet)o;
        int value = fbs.freq.compareTo(freq);
        LogWriter.logger.finest("Returning value :'"+value+"'");
        LogWriter.logger.exiting("FreqBitSet", "compareTo");
        return value;
    }
}
```

CommonTools class:

```
/*************************************************************
 * Copyright (C) 2003 Pekka Partanen <pekka.partanen@iki.fi>  *
 * All rights reserved.                                       *
 *                                                            *
 * This program is free software; you can redistribute it     *
 * and/or modify it under the terms of the GNU General Public *
 * License. See the file COPYRIGHT for more information.      *
 *************************************************************/
package com.aul.tft;

/**
 * General tools for other classes.
 *
 * @author      <a href="pekka.partanen@iki.fi">Pekka Partanen</a>
 * @version     1.0
 * @since       1.0
 */

public class CommonTools {

    /**
     * Suppressed default constructor.
     */
    private CommonTools() {}

    /**
     * Handles the dynamic content of the user interface
     * description. The user interface description file can have
     * fields in angle brackets to represent dynamic content.
     * <p>
     * Method is used to convert the changing parameter values to
     * abstracted constant values. The user interface description
     * fields that contain angle brackets are converted to a
     * default value. In the same way, weblog entries that have
     * '=.*' values in their query part are converted to the same
     * default value.
     *
     * @param  key HTTP query line for conversion. Could be from
     * the user interface description file or from the weblog
     * @return key converted HTTP query line
     * @since  1.0
     */
    public static String convertRequestlineToName(String key) {

        // For files in the UI element description
        key =
            key.replaceAll("\\=<[0-9a-zA-Z[-][+][%] ]*>", "FFF");

        // Following line handles the parameters' values in the
        // real HTTP request
        key = (key.replaceAll("\\=[0-9a-zA-Z[-][+][%] ]*",
                              "FFF")).trim();
        return key;
```

```
        }
}
```

Appendix B

Link Structure of MDS/SAS 5.0

```
Main -> {
  /sas5/navigation_servlet #Main Menu# "MDS/SAS 5.0"
  /sas5/index.jsp #Main Menu# "Logout"
  /sas5/navigation_servlet/showEmptyNetwork #Main Menu# "Network"
  /sas5/navigation_servlet/showEmptyMonitoring #Main Menu# "Monitoring"
  /sas5/navigation_servlet/showEmptyOperationalConfiguration #Main Menu#
       "Operational_Configuration"
  /sas5/navigation_servlet/showEmptyMaintenance #Main Menu# "Maintenance"
  /sas5/navigation_servlet/showEmptyUserMaintenance #Main Menu#
       "User_Maintenance"
}

Network -> {
  /sas5/network_servlet #Network menu# "Network_Model"
  /sas5/ne_types_servlet #Network menu# "NE_Types"
  /sas5/ne_interface_types_servlet #Network menu# "NE_Interface_Types"
  /sas5/connection_types_servlet #Network menu# "Connection_Types"
  /sas5/default_ne_parameters_servlet #Network menu# "Default_NE_Parameters"
}

Monitoring -> {
  /sas5/requests_servlet #Monitoring menu# "Requests"
  /sas5/tasks_servlet?entry=<maybe yes> #Monitoring menu# "Tasks"
  /sas5/total_statistics_servlet #Monitoring menu# "MDSSAS_Total_Statistics"
  /sas5/client_system_statistics_servlet #Monitoring menu#
       "MDSSAS_Client_Systems"
  /sas5/ne_statistics_servlet #Monitoring menu# "NE_Statistics"
  /sas5/core_statistics_servlet #Monitoring menu# "MDSSAS_Core_Statistics"
  /sas5/maintenance_log_servlet #Monitoring menu# "Maintenance_Log"
}

Operational_Configuration -> {
  /sas5/grc_servlet #Operational Configuration menu# "GRC_Parameters"
  /sas5/service_modules_servlet #Operational Configuration menu#
       "Service_Modules"
}
```

```
Maintenance -> {
  /sas5/system_status_servlet #Maintenance menu# "System_Status"
  /sas5/export_import_servlet/showImport #Maintenance menu#
       "Configuration_Importing"
  /sas5/reports_servlet #Maintenance menu# "Reports"
}

User_Maintenance -> {
  /sas5/ui_profile_servlet #User Maintenance menu# "UI_Profiles"
  /sas5/ui_user_servlet/showUsers #User Maintenance menu# "UI_Users"
  /sas5/ui_user_servlet/showPassword?id=<username> #User Maintenance menu#
       "Set_UI_Password"
  /sas5/client_user_servlet/showUsers #User Maintenance menu# "Client_Users"
}

Network_Model -> {
  /sas5/NetworkModel.jsp?REFRESH.RATE=<60> #Network Model submenu#
        "1_min"
  /sas5/NetworkModel.jsp?REFRESH.RATE=<120> #Network Model submenu#
        "2_min"
  /sas5/NetworkModel.jsp?REFRESH.RATE=<600> #Network Model submenu#
        "10_min"
  /sas5/NetworkModel.jsp?REFRESH.RATE=<0> #Network Model submenu#
        "no_refresh"
  /sas5/network_servlet/updateView/?element=<NE> #Network Model submenu#
       "Plus_Symbol_Network_Model"
  /sas5/network_servlet/showNeDetails?neId=<NE> #Network Model submenu#
       "NE_Details"
  /sas5/network_servlet/showNeAdding?newNe=<yes>#Network Model submenu#
       "Add_NE"
  /sas5/network_servlet/showConnectionAdding?newConnection=<yes>
       #Network_Model_submenu# "Add_Connection"
}

NE_Types -> {
  /sas5/ne_types_servlet/showNETypeDetailsReadOnly?id=<NE Type>
       #NE Types submenu# "NE_Type_Details"
  /sas5/ne_types_servlet/showNETypeDetails?id=<NE Type> #NE Types submenu#
       "Modify_NE_Type"
  /sas5/ne_types_servlet/showNETypeDeleteConfirmation?id=<parameter>
       #NE Types submenu# "Delete_Parameter"
  /sas5/ne_types_servlet/createNEType #NE Types submenu# "Add_New_NE_Types"
}

NE_Interface_Types -> {
  /sas5/ne_interface_types_servlet/newNEInterfaceType
       #NE Interface Types submenu# "Add_New_NE_Interface_Types"
  /sas5/ne_interface_types_servlet/modifyNEInterfaceType?type=<type>
       #NE Interface Types submenu# "Modify_NE_Interface_Types"
  /sas5/ne_interface_types_servlet/deleteConfirmation?type=<type>
       #NE Interface Types submenu# "Delete_NE_Interface_Type"
}

Connection_Types -> {
  /sas5/connection_types_servlet/modifyConnectionType?type=<type>
```

```
        #Connection Types submenu# "Modify_Connection_Types"
  /sas5/connection_types_servlet/deleteConfirmation?type=<type>
        #Connection Types submenu# "Delete_Connection_Type"
  /sas5/connection_types_servlet/newConnectionType
        #Connection Types submenu# "Add_New_Connection_Types"
}

Default_NE_Parameters -> {
  /sas5/default_ne_parameters_servlet/modifyDefaultParameter?id=
        <parameter> #Default NE Parameters submenu#
        "Modify_Default_NE_Parameter"
  /sas5/default_ne_parameters_servlet/deleteConfirmation?id=
        <parameter> #Default NE Parameters submenu#
        "Delete_NE_Parameter"
  /sas5/default_ne_parameters_servlet/newDefaultParameter
        #Default NE Parameters submenu# "Add_New_Default_NE_Parameters"
}

Requests -> {
  /sas5/Requests.jsp?searchForm=<false> #Requests submenu#
        "HIDE_SHOW_SEARCH_FORM_Requests"
  /sas5/requests_servlet/selectAllRequests #Requests submenu#
        "Select_All_Requests"
  /sas5/requests_servlet/unselectAllRequests #Requests submenu#
        "Unselect_All_Requests"
  /sas5/requests_servlet/showRequestDetails?requestId=<requestID>
        #Requests submenu# "Request_Details"
}

Tasks -> {
  /sas5/tasks_servlet/toggleSearchForm?taskSearchForm=<false>
        #Tasks submenu# "HIDE_SHOW_SEARCH_FORM_Tasks"
  /sas5/tasks_servlet/selectAllTasks #Tasks submenu# "Select_All_Tasks"
  /sas5/tasks_servlet/unselectAllTasks #Tasks submenu# "Unselect_All_Tasks"
  /sas5/tasks_servlet/showTaskDetails?taskId=<task ID>&requestId=
        <request ID> #Tasks submenu# "Task_Details"
}

NE_Statistics -> {
  /sas5/ne_statistics_servlet/showNEDetailedStatistics
        #NE Statistics submenu# "More_Detailed_Table"
  /sas5/ne_statistics_servlet/showNEStatistics #NE Statistics submenu#
        "Concise_Table"
}

MDSSAS_Core_Statistics -> {
  /sas5/core_statistics_servlet/search?layer=<0>
        #MDS/SAS Core Statistics submenu# "Client_System_Layer"
  /sas5/core_statistics_servlet/search?layer=<1>
        #MDS/SAS Core Statistics submenu# "Request_Processing_Layer"
  /sas5/core_statistics_servlet/search?layer=<2>
        #MDS/SAS Core Statistics submenu# "Service_Module_Layer"
  /sas5/core_statistics_servlet/search?layer=<3>
        #MDS/SAS Core Statistics submenu# "Task_Engine_Layer"
}
```

```
GRC_Parameters -> {
  /sas5/grc_servlet/showParameters?<id=comptel.cs.nemo>
       #GRC Parameters submenu# "comptel_cs_nemo"
  /sas5/grc_servlet/showParameters?id=<comptel.sas>
       #GRC Parameters submenu# "comptel_sas"
  /sas5/grc_servlet/showParameters?id=<comptel.sas.gui>
       #GRC Parameters submenu# "comptel_sas_gui"
  /sas5/grc_servlet/showParameters?id=<comptel.sas.re>
       #GRC Parameters submenu# "comptel_sas_re"
  /sas5/grc_servlet/showParameters?id=<comptel.sas.te>
       #GRC Parameters submenu# "comptel_sas_te"
  /sas5/grc_servlet/sectionDeleteConfirmation?id=<name of section>
       #GRC Parameters submenu# "Delete_GRC_Section"
  /sas5/grc_servlet/newSection #GRC Parameters submenu#
       "Add_New_GRC_Parameter"
}

Service_Modules -> {
  /sas5/service_modules_servlet?&broker.pluginId=
       <id of plug-in>&broker.request=<details>
       #Service Modules submenu# "Routing_Service"
}

System_Status -> {
  /sas5/SystemStatus.jsp?refreshRate=<5> #System Status#
       "5_s_refresh_rate_"
  /sas5/SystemStatus.jsp?refreshRate=<20> #System Status#
       "20_s_refresh_rate"
  /sas5/SystemStatus.jsp?refreshRate=<60> #System Status#
       "60_s_refresh_rate’"
  /sas5/SystemStatus.jsp?refreshRate=<0> #System Status#
       "no_refresh’"
  /sas5/component_log_servlet?prefix=<NM> #System Status submenu# "Nemo"
  /sas5/component_log_servlet?prefix=<TE> #System Status submenu#
       "TaskEngine"
  /sas5/component_log_servlet?prefix=<> #System Status submenu#
       "RequestEngine"
  /sas5/component_log_servlet?prefix=<UI> #System Status submenu#
       "User_Interface"
}

Reports -> {
  /sas5/reports_servlet/showParameterSetting?name=<Database usage>
       #Reports submenu# "Database_Usage_Reports"
  /sas5/reports_servlet/showParameterSetting?name=<GRC> #Reports submenu#
       "GRC_Reports"
  /sas5/reports_servlet/showParameterSetting?name=<Network model>
       #Reports submenu# "Network_Model_Reports"
  /sas5/reports_servlet/showParameterSetting?name=<Request details>
       #Reports submenu# "Request_Details_Reports"
  /sas5/reports_servlet/showParameterSetting?name=<Request summary>
       #Reports submenu# "Request_Summary_Reports"
  /sas5/reports_servlet/showParameterSetting?name=<Requests>
       #Reports submenu# "Requests_Reports"
```

```
  /sas5/reports_servlet/showParameterSetting?name=<Service modules>
       #Reports submenu# "Service_Modules_Reports"
  /sas5/reports_servlet/showParameterSetting?name=<Task summary>
       #Reports submenu# "Task_Summary_Reports"
  /sas5/reports_servlet/showParameterSetting?name=<Summary for NE>
       #Reports submenu# "Task_Summary_For_NE_Reports"
  /sas5/reports_servlet/showParameterSetting?name=<Summary for NE type>
       #Reports submenu# "Task_Summary_For_NE_Type_Reports"
  /sas5/reports_servlet/showParameterSetting?name=<User management>
       #Reports submenu# "User_Management_Reports"
}

UI_Profiles -> {
  /sas5/ui_profile_servlet/showProfileDetails?name=<profile>
       #UI Profiles submenu# "UI_profile_name"
  /sas5/ui_profile_servlet/modifyProfileFromList?name=<profile>
       #UI Profiles submenu# "Modify_Profile_Details"
  /sas5/ui_profile_servlet/deleteConfRequest?id=<profile>&caller=
       <profiles> #UI Profiles submenu# "Delete_UI_profile"
  /sas5/ui_profile_servlet/newProfile #UI Profiles submenu#
       "Add_New_UI_Profiles"
}

UI_Users -> {
  /sas5/ui_user_servlet/showUserDetails?name=<GUIuser> #UI Users submenu#
       "UI_User_Details"
  /sas5/ui_user_servlet/modifyUserFromList?name=<GUIuser>
       #UI Users submenu# "Modify_UI_User_Details"
  /sas5/ui_user_servlet/deleteConfRequest?id=<user>&userCaller=
       <caller> #UI Users submenu# "Delete_UI_User"
  /sas5/ui_user_servlet/newUser #UI Users submenu# "Add_New_UI_Users"
}

Client_Users -> {
  /sas5/client_user_servlet/showUserDetails?name=<bss5>
       #Client Users submenu# "Username_Client_Users"
  /sas5/client_user_servlet/modifyUserFromList?name=<username>
       #Client Users submenu# "Modify_Client_User_Details"
  /sas5/client_user_servlet/deleteConfRequest?id=<username>&userCaller=
       <caller> #Client Users submenu# "Delete_Client_User"
  /sas5/client_user_servlet/newUser #Client Users submenu#
       "Add_New_Client_Users"
  /sas5/client_user_servlet/modifyUserFromReadOnly?name=<username>
       #Client Users submenu# "Modify_Client_Users"
}

Plus_Symbol_Network_Model -> {
  /sas5/network_servlet/showConnectionDetails?conId=<NE>&sourceNe=
       <NE>&targetNe=<target NE> #'+' symbol subsubmenu#
       "Connection_Details "
}

NE_Details -> {
  /sas5/network_servlet/showNeModify?neId=<NE> #NE Details submenu#
       "Modify_NE_Details"
```

```
  /sas5/network_servlet/lockNetworkElement #NE Details submenu#
       "Lock_NE_Details"
  /sas5/network_servlet/unlockNetworkElement #NE Details submenu#
       "Unlock_NE_Details"
  /sas5/network_servlet/showNeDeleteConfirmation #NE Details submenu#
       "Delete_NE_Details"
  /sas5/network_servlet/showNeDeleteConfirmation #NE Details submenu#
       "Duplicate_NE_Details"
  /sas5/network_servlet/showNeParameterModify?paramName=
       <NE parameter> #NE Details submenu# "Modify_NE_Parameter"
  /sas5/network_servlet/showNeParamDelConfirmation?paramName=
       <parameter> #NE Details submenu# "Delete_NE_parameter"
  /sas5/network_servlet/showNeParameterAdding?newParam=<yes>
       #NE Details submenu# "Add_New_NE_Details"
}

NE_Type_Details -> {
  /sas5/ne_types_servlet/showNETypeDetails?id=<NE>
       #NE Type Details subsubmenu# "Modify_NE_Type_Details"
}

Request_Details -> {
  /sas5/requests_servlet/showRequestParameters
       #Request Details subsubmenu# "Parameters_Request_Details"
  /sas5/requests_servlet/showTaskDetails?taskId=<task ID number>
       #Request Details <requestID> subsubmenu# "Task_ID_Number"
}

Task_Details -> {
  /sas5/tasks_servlet/showTaskParameters?taskId=
       <number of taskId>&requestId=<number of requestId>
       #Task Details subsubmenu# "Parameters_Task_Details"
  /sas5/log_servlet/searchIOLog?taskId=<number of taskId>&neId=
       <Id of NE>&startDate=<startdate>&endDate=<enddate>&channel=
       <number of channel> #Task Details subsubmenu# "IO_Log"
  /sas5/log_servlet/searchMMLLog?&taskId=<number of taskId>&neId=
       <Id of NE>&startDate=<startdate>&endDate=<enddate>&channel=
       <number of channel> #Task Details subsubmenu# "MML_Log"
  /sas5/log_servlet/searchErrorLog?&taskId=<number of taskId>&neId=
       <Id of NE>&startDate=<startdate>&endDate=<enddate>&channel=
       <number of channel> #Task Details subsubmenu# "Error_Log"
}

comptel_cs_nemo -> {
  /sas5/grc_servlet/modifyParameter?id=<parameter>
       #GRC Parameters subsubmenu# "Modify_GRC_Parameters"
  /sas5/grc_servlet/paramDeleteConfirmation?id=<parameter>
       #GRC Parameters subsubmenu# "Delete_GRC_Parameter"
  /sas5/grc_servlet/newParameter #GRC Parameters subsubmenu#
       "Add_new_GRC_Parameter"
}

comptel_sas -> {
  /sas5/grc_servlet/modifyParameter?id=<name of parameter>
       #GRC Parameter subsubmenu# "Modify_GRC_Parameters"
```

```
  /sas5/grc_servlet/paramDeleteConfirmation?id=<parameter>
       #GRC Parameters subsubmenu# "Delete_GRC_Parameter"
  /sas5/grc_servlet/newParameter #GRC Parameters subsubmenu#
       "Add_new_GRC_Parameter"
}

comptel_sas_gui
  /sas5/grc_servlet/modifyParameter?id=<parameter>
       #GRC Parameters subsubmenu# "Modify_GRC_Parameters"
  /sas5/grc_servlet/paramDeleteConfirmation?id=<parameter>
       #GRC Parameters subsubmenu# "Delete_GRC_Parameter"
  /sas5/grc_servlet/newParameter #GRC Parameters subsubmenu#
       "Add_new_GRC_Parameter"
}

comptel_sas_re -> {
  /sas5/grc_servlet/modifyParameter?id=<parameter>
       #GRC Parameters subsubmenu# "Modify_GRC_Parameters"
  /sas5/grc_servlet/paramDeleteConfirmation?id=<parameter>
       #GRC Parameters subsubmenu# "Delete_GRC_Parameter"
  /sas5/grc_servlet/newParameter #GRC Parameters subsubmenu#
       "Add_new_GRC_Parameter"
}

comptel_sas_te -> {
  /sas5/grc_servlet/modifyParameter?id=<parameter>
       #GRC Parameters subsubmenu# "Modify_GRC_Parameters"
  /sas5/grc_servlet/paramDeleteConfirmation?id=<parameter>
       #GRC Parameters subsubmenu# "Delete_GRC_Parameter"
  /sas5/grc_servlet/newParameter #GRC Parameters subsubmenu#
       "Add_new_GRC_Parameter"
}

Routing_Service -> {
  /sas5/service_modules_servlet?&broker.pluginGuild=
       <service>&broker.request=<request>&action=<action>
       #<service module> subsubmenu# "List_Number_Ranges"
}

UI_User_Details -> {
  /sas5/ui_user_servlet/modifyUserFromReadOnly?name=<GUIuser>
       #UI User Details subsubmenu# "Modify_UI_User_Details"
}

Client_User_Details -> {
  /sas5/client_user_servlet/modifyUserFromReadOnly?name=
       <username> #Client User Details subsubmenu#
       "Modify_Client_User_Details"
}

Connection_Details -> {
  /sas5/network_servlet/showConnectionModify?conId=<NE>
       #Connection Details subsubsubmenu# "Modify_Connection_Details"
  /sas5/network_servlet/lockConnection
       #Connection Details subsubsubmenu# "Lock_Connection_Details"
```

```
  /sas5/network_servlet/unlockConnection
       #Connection Details subsubsubmenu# "Unlock_Connection_Details"
  /sas5/network_servlet/showConnectionDeleteConfirmation
       #Connection Details subsubsubmenu# "Delete_Connection_Details"
  /sas5/network_servlet/showConnectionAdding?copy=<yes>
       #Connection Details subsubsubmenu# "Duplicate_Connection_Details"
}

List_Number_Ranges -> {
  /sas5/service_modules_servlet?&broker.pluginGuild=
       <service>&broker.request=<request>&action=<printEditRange>
       #List Number Ranges subsubsubmenu# "Edit_Number_Range"
  /sas5/service_modules_servlet?&broker.pluginGuild=
       <service>&broker.request=<request>&action=<deleteRange>&ra
       #List Number Ranges subsubsubmenu# "Delete_Number_Range"
  /sas5/service_modules_servlet?&broker.pluginGuild=
       <service>&broker.request=<request>&action=<printCreateRange>
       #List Number Ranges subsubsubmenu# "Create_New_Number_Range"
}
```

Appendix C

Weblog Excerpt

The following weblog was collected from a single session. The IP-address is masked for security. The result of filtration is presented after the weblog.

```
x.x.x.160 - - [13/Mar/2003:12:43:41 2000] "GET /sas5/index.jsp HTTP/1.1" 200 3592
x.x.x.160 - - [13/Mar/2003:12:43:41 2000] "GET /sas5/styles/sas5_ie.css HTTP/1.1" 200 16665
x.x.x.160 - - [13/Mar/2003:12:43:41 2000] "GET /sas5/images/dot.gif HTTP/1.1" 200 43
x.x.x.160 - - [13/Mar/2003:12:43:41 2000] "GET /sas5/images/comptel.gif HTTP/1.1" 200 1862
x.x.x.160 - - [13/Mar/2003:12:43:46 2000] "POST /sas5/ui_user_servlet HTTP/1.1" 302 654
x.x.x.160 - - [13/Mar/2003:12:43:46 2000] "GET /sas5/navigation_servlet/show?
                                           pageid=1047552226441 HTTP/1.1" 302 654
x.x.x.160 - - [13/Mar/2003:12:43:46 2000] "GET /sas5/Frames.jsp?pageid=1047552226459
                                           HTTP/1.1" 200 1292
x.x.x.160 - - [13/Mar/2003:12:43:46 2000] "GET /sas5/Header.jsp HTTP/1.1" 200 1122
x.x.x.160 - - [13/Mar/2003:12:43:46 2000] "GET /sas5/Menu.jsp HTTP/1.1" 200 1306
x.x.x.160 - - [13/Mar/2003:12:43:46 2000] "GET /sas5/EmptyMenu.jsp HTTP/1.1" 200 934
x.x.x.160 - - [13/Mar/2003:12:43:46 2000] "GET /sas5/Empty.html HTTP/1.1" 200 289
x.x.x.160 - - [13/Mar/2003:12:43:46 2000] "GET /sas5/Welcome.jsp HTTP/1.1" 200 491
x.x.x.160 - - [13/Mar/2003:12:43:46 2000] "GET /sas5/Footer.jsp HTTP/1.1" 200 950
x.x.x.160 - - [13/Mar/2003:12:43:46 2000] "GET /sas5/images/dot.gif HTTP/1.1" 200 43
x.x.x.160 - - [13/Mar/2003:12:43:46 2000] "GET /sas5/styles/sas5_ie.css HTTP/1.1" 200 16665
x.x.x.160 - - [13/Mar/2003:12:43:46 2000] "GET /sas5/images/comptel.gif HTTP/1.1" 200 1862
x.x.x.160 - - [13/Mar/2003:12:43:46 2000] "GET /sas5/images/menuBG.gif HTTP/1.1" 200 138
x.x.x.160 - - [13/Mar/2003:12:43:48 2000] "GET /sas5/navigation_servlet/showEmptyNetwork
                                           HTTP/1.1" 302 654
x.x.x.160 - - [13/Mar/2003:12:43:48 2000] "GET /sas5/Frames.jsp?pageid=1047552228058
                                           HTTP/1.1" 200 1312
x.x.x.160 - - [13/Mar/2003:12:43:48 2000] "GET /sas5/Header.jsp HTTP/1.1" 200 1122
x.x.x.160 - - [13/Mar/2003:12:43:48 2000] "GET /sas5/NetworkNavigation.jsp HTTP/1.1" 200 1313
x.x.x.160 - - [13/Mar/2003:12:43:48 2000] "GET /sas5/NetworkMenu.jsp HTTP/1.1" 200 1525
x.x.x.160 - - [13/Mar/2003:12:43:48 2000] "GET /sas5/EmptyNetwork.jsp HTTP/1.1" 200 87
x.x.x.160 - - [13/Mar/2003:12:43:48 2000] "GET /sas5/Empty.html HTTP/1.1" 200 289
x.x.x.160 - - [13/Mar/2003:12:43:48 2000] "GET /sas5/Footer.jsp HTTP/1.1" 200 950
x.x.x.160 - - [13/Mar/2003:12:43:48 2000] "GET /sas5/images/dot.gif HTTP/1.1" 200 43
x.x.x.160 - - [13/Mar/2003:12:43:48 2000] "GET /sas5/images/comptel.gif HTTP/1.1" 200 1862
x.x.x.160 - - [13/Mar/2003:12:43:48 2000] "GET /sas5/styles/sas5_ie.css HTTP/1.1" 200 16665
x.x.x.160 - - [13/Mar/2003:12:43:48 2000] "GET /sas5/images/menuBG.gif HTTP/1.1" 200 138
x.x.x.160 - - [13/Mar/2003:12:43:53 2000] "GET /sas5/network_servlet HTTP/1.1" 302 654
x.x.x.160 - - [13/Mar/2003:12:43:53 2000] "GET /sas5/NetworkModel.jsp?pageid=1047552233518
                                           HTTP/1.1" 200 7688
x.x.x.160 - - [13/Mar/2003:12:43:53 2000] "GET /sas5/styles/sas5_ie.css HTTP/1.1" 200 16665
x.x.x.160 - - [13/Mar/2003:12:43:53 2000] "GET /sas5/help/RoboHelp_CSH.js HTTP/1.1" 200 6709
x.x.x.160 - - [13/Mar/2003:12:43:53 2000] "GET /sas5/images/workstation.gif HTTP/1.1" 200 1290
```

```
x.x.x.160 - - [13/Mar/2003:12:43:53 2000] "GET /sas5/images/plus.gif HTTP/1.1" 200 927
x.x.x.160 - - [13/Mar/2003:12:43:53 2000] "GET /sas5/images/managed_ne.gif HTTP/1.1" 200 1025
x.x.x.160 - - [13/Mar/2003:12:43:55 2000] "GET /sas5/network_servlet/showNeDetails?neId=OMC13
                                           HTTP/1.1" 302 654
x.x.x.160 - - [13/Mar/2003:12:43:55 2000] "GET /sas5/NeDetails.jsp?pageid=1047552235448
                                           HTTP/1.1" 200 5893
x.x.x.160 - - [13/Mar/2003:12:44:02 2000] "GET /sas5/network_servlet/showNetworkModel
                                           HTTP/1.1" 302 654
x.x.x.160 - - [13/Mar/2003:12:44:02 2000] "GET /sas5/NetworkModel.jsp?pageid=1047552242217
                                           HTTP/1.1" 200 7688
x.x.x.160 - - [13/Mar/2003:12:44:02 2000] "GET /sas5/images/workstation.gif HTTP/1.1" 200 1290
x.x.x.160 - - [13/Mar/2003:12:44:02 2000] "GET /sas5/images/managed_ne.gif HTTP/1.1" 200 1025
x.x.x.160 - - [13/Mar/2003:12:44:02 2000] "GET /sas5/images/plus.gif HTTP/1.1" 200 927
x.x.x.160 - - [13/Mar/2003:12:44:04 2000] "GET /sas5/network_servlet/updateView?element=OMC13
                                           HTTP/1.1" 302 654
x.x.x.160 - - [13/Mar/2003:12:44:04 2000] "GET /sas5/NetworkModel.jsp?pageid=1047552244042
                                           HTTP/1.1" 200 8173
x.x.x.160 - - [13/Mar/2003:12:44:04 2000] "GET /sas5/images/plus.gif HTTP/1.1" 200 927
x.x.x.160 - - [13/Mar/2003:12:44:04 2000] "GET /sas5/images/top_begin.gif HTTP/1.1" 200 907
x.x.x.160 - - [13/Mar/2003:12:44:04 2000] "GET /sas5/images/workstation.gif HTTP/1.1" 200 1290
x.x.x.160 - - [13/Mar/2003:12:44:04 2000] "GET /sas5/images/minus.gif HTTP/1.1" 200 931
x.x.x.160 - - [13/Mar/2003:12:44:04 2000] "GET /sas5/images/managed_ne.gif HTTP/1.1" 200 1025
x.x.x.160 - - [13/Mar/2003:12:44:06 2000] "GET /sas5/network_servlet/showConnectionDetails?
                          conId=mds1_OMC13_399&sourceNe=mds1&targetNe=OMC13 HTTP/1.1" 302 654
x.x.x.160 - - [13/Mar/2003:12:44:06 2000] "GET /sas5/ConnectionDetails.jsp?pageid=1047552246295
                                           HTTP/1.1" 200 2835
x.x.x.160 - - [13/Mar/2003:12:44:18 2000] "GET /sas5/network_servlet/showNetworkModel
                                           HTTP/1.1" 302 654
x.x.x.160 - - [13/Mar/2003:12:44:18 2000] "GET /sas5/NetworkModel.jsp?pageid=1047552258374
                                           HTTP/1.1" 200 7688
x.x.x.160 - - [13/Mar/2003:12:44:18 2000] "GET /sas5/images/managed_ne.gif HTTP/1.1" 200 1025
x.x.x.160 - - [13/Mar/2003:12:44:18 2000] "GET /sas5/images/plus.gif HTTP/1.1" 200 927
x.x.x.160 - - [13/Mar/2003:12:44:18 2000] "GET /sas5/images/workstation.gif HTTP/1.1" 200 1290
x.x.x.160 - - [13/Mar/2003:12:44:19 2000] "GET /sas5/network_servlet/showNeDetails?neId=OMC13
                                           HTTP/1.1" 302 654
x.x.x.160 - - [13/Mar/2003:12:44:19 2000] "GET /sas5/NeDetails.jsp?pageid=1047552259963
                                           HTTP/1.1" 200 5893
x.x.x.160 - - [13/Mar/2003:12:44:28 2000] "GET /sas5/ne_types_servlet HTTP/1.1" 302 654
x.x.x.160 - - [13/Mar/2003:12:44:28 2000] "GET /sas5/NeTypes.jsp?pageid=1047552268074
                                           HTTP/1.1" 200 4421
x.x.x.160 - - [13/Mar/2003:12:44:30 2000] "GET /sas5/ne_interface_types_servlet
                                           HTTP/1.1" 200 1946
x.x.x.160 - - [13/Mar/2003:12:44:31 2000] "GET /sas5/connection_types_servlet HTTP/1.1" 200 2546
x.x.x.160 - - [13/Mar/2003:12:44:33 2000] "GET /sas5/default_ne_parameters_servlet
                                           HTTP/1.1" 200 6374
x.x.x.160 - - [13/Mar/2003:12:44:37 2000] "GET /sas5/navigation_servlet/showEmptyMaintenance
                                           HTTP/1.1" 302 654
x.x.x.160 - - [13/Mar/2003:12:44:37 2000] "GET /sas5/Frames.jsp?pageid=1047552277928
                                           HTTP/1.1" 200 1324
x.x.x.160 - - [13/Mar/2003:12:44:37 2000] "GET /sas5/Header.jsp HTTP/1.1" 200 1122
x.x.x.160 - - [13/Mar/2003:12:44:37 2000] "GET /sas5/MaintenanceNavigation.jsp
                                           HTTP/1.1" 200 1311
x.x.x.160 - - [13/Mar/2003:12:44:37 2000] "GET /sas5/MaintenanceMenu.jsp HTTP/1.1" 200 1400
x.x.x.160 - - [13/Mar/2003:12:44:37 2000] "GET /sas5/EmptyMaintenance.jsp HTTP/1.1" 200 103
x.x.x.160 - - [13/Mar/2003:12:44:37 2000] "GET /sas5/Empty.html HTTP/1.1" 200 289
x.x.x.160 - - [13/Mar/2003:12:44:37 2000] "GET /sas5/Footer.jsp HTTP/1.1" 200 950
x.x.x.160 - - [13/Mar/2003:12:44:37 2000] "GET /sas5/images/dot.gif HTTP/1.1" 200 43
x.x.x.160 - - [13/Mar/2003:12:44:37 2000] "GET /sas5/styles/sas5_ie.css HTTP/1.1" 200 16665
x.x.x.160 - - [13/Mar/2003:12:44:37 2000] "GET /sas5/images/comptel.gif HTTP/1.1" 200 1862
x.x.x.160 - - [13/Mar/2003:12:44:37 2000] "GET /sas5/images/menuBG.gif HTTP/1.1" 200 138
x.x.x.160 - - [13/Mar/2003:12:44:39 2000] "GET /sas5/system_status_servlet HTTP/1.1" 302 654
x.x.x.160 - - [13/Mar/2003:12:44:39 2000] "GET /sas5/SystemStatus.jsp?pageid=1047552279917
                                           HTTP/1.1" 200 4471
x.x.x.160 - - [13/Mar/2003:12:44:39 2000] "GET /sas5/help/RoboHelp_CSH.js HTTP/1.1" 200 6709
x.x.x.160 - - [13/Mar/2003:12:44:39 2000] "GET /sas5/styles/sas5_ie.css HTTP/1.1" 200 16665
x.x.x.160 - - [13/Mar/2003:12:44:42 2000] "GET
                 /sas5/navigation_servlet/showEmptyOperationalConfiguration HTTP/1.1" 302 654
x.x.x.160 - - [13/Mar/2003:12:44:42 2000] "GET /sas5/Frames.jsp?pageid=1047552282538
                                           HTTP/1.1" 200 1349
```

```
x.x.x.160 - - [13/Mar/2003:12:44:42 2000] "GET /sas5/Header.jsp HTTP/1.1" 200 1122
x.x.x.160 - - [13/Mar/2003:12:44:42 2000] "GET /sas5/OperConfigNavigation.jsp HTTP/1.1" 200 1313
x.x.x.160 - - [13/Mar/2003:12:44:42 2000] "GET /sas5/OperationalConfigurationMenu.jsp
                                           HTTP/1.1" 200 1260
x.x.x.160 - - [13/Mar/2003:12:44:42 2000] "GET /sas5/EmptyOperationalConfiguration.jsp
                                           HTTP/1.1" 200 103
x.x.x.160 - - [13/Mar/2003:12:44:42 2000] "GET /sas5/Empty.html HTTP/1.1" 200 289
x.x.x.160 - - [13/Mar/2003:12:44:42 2000] "GET /sas5/Footer.jsp HTTP/1.1" 200 950
x.x.x.160 - - [13/Mar/2003:12:44:42 2000] "GET /sas5/styles/sas5_ie.css HTTP/1.1" 200 16665
x.x.x.160 - - [13/Mar/2003:12:44:42 2000] "GET /sas5/images/dot.gif HTTP/1.1" 200 43
x.x.x.160 - - [13/Mar/2003:12:44:42 2000] "GET /sas5/images/comptel.gif HTTP/1.1" 200 1862
x.x.x.160 - - [13/Mar/2003:12:44:42 2000] "GET /sas5/images/menuBG.gif HTTP/1.1" 200 138
x.x.x.160 - - [13/Mar/2003:12:44:44 2000] "GET /sas5/grc_servlet HTTP/1.1" 302 654
x.x.x.160 - - [13/Mar/2003:12:44:44 2000] "GET /sas5/GRCSections.jsp?pageid=1047552284126
                                           HTTP/1.1" 200 1843
x.x.x.160 - - [13/Mar/2003:12:44:44 2000] "GET /sas5/help/RoboHelp_CSH.js HTTP/1.1" 200 6709
x.x.x.160 - - [13/Mar/2003:12:44:44 2000] "GET /sas5/styles/sas5_ie.css HTTP/1.1" 200 16665
x.x.x.160 - - [13/Mar/2003:12:44:45 2000] "GET /sas5/grc_servlet/showParameters?id=
                                           comptel.cs.nemo HTTP/1.1" 302 654
x.x.x.160 - - [13/Mar/2003:12:44:45 2000] "GET /sas5/GRCParameters.jsp?pageid=1047552285178
                                           HTTP/1.1" 200 2164
x.x.x.160 - - [13/Mar/2003:12:44:47 2000] "GET /sas5/GRCSections.jsp HTTP/1.1" 200 1843
x.x.x.160 - - [13/Mar/2003:12:44:47 2000] "GET /sas5/grc_servlet/showParameters?id=
                                           comptel.sas HTTP/1.1" 302 654
x.x.x.160 - - [13/Mar/2003:12:44:47 2000] "GET /sas5/GRCParameters.jsp?pageid=1047552288742
                                           HTTP/1.1" 200 1715
x.x.x.160 - - [13/Mar/2003:12:44:49 2000] "GET /sas5/GRCSections.jsp HTTP/1.1" 200 1843
x.x.x.160 - - [13/Mar/2003:12:44:49 2000] "GET /sas5/grc_servlet/showParameters?
                                           id=comptel.sas.gui HTTP/1.1" 302 654
x.x.x.160 - - [13/Mar/2003:12:44:51 2000] "GET /sas5/GRCParameters.jsp?pageid=1047552290976
                                           HTTP/1.1" 200 4576
x.x.x.160 - - [13/Mar/2003:12:44:57 2000] "GET /sas5/grc_servlet HTTP/1.1" 302 654
x.x.x.160 - - [13/Mar/2003:12:44:57 2000] "GET /sas5/GRCSections.jsp?pageid=1047552297863
                                           HTTP/1.1" 200 1843
x.x.x.160 - - [13/Mar/2003:12:44:59 2000] "GET /sas5/grc_servlet/showParameters?id=
                                           comptel.sas.re HTTP/1.1" 302 654
x.x.x.160 - - [13/Mar/2003:12:44:59 2000] "GET /sas5/GRCParameters.jsp?pageid=1047552299304
                                           HTTP/1.1" 200 7468
x.x.x.160 - - [13/Mar/2003:12:45:09 2000] "GET /sas5/GRCSections.jsp HTTP/1.1" 200 1843
x.x.x.160 - - [13/Mar/2003:12:45:10 2000] "GET /sas5/grc_servlet/showParameters?id=
                                           comptel.sas.te HTTP/1.1" 302 654
x.x.x.160 - - [13/Mar/2003:12:45:10 2000] "GET /sas5/GRCParameters.jsp?pageid=1047552310676
                                           HTTP/1.1" 200 4708
x.x.x.160 - - [13/Mar/2003:12:47:52 2000] "GET /sas5/ui_user_servlet/logout HTTP/1.1" 302 654
```

The result of filtering the weblog against the hierarchical user interface description (Appendix B) is below:

```
/sas5/navigation_servlet/showEmptyNetwork
/sas5/network_servlet
/sas5/network_servlet/showNeDetails?neIdFFF
/sas5/network_servlet/updateView?elementFFF
/sas5/network_servlet/showConnectionDetails?conIdFFF&sourceNeFFF&targetNeFFF
/sas5/network_servlet/showNeDetails?neIdFFF
/sas5/ne_types_servlet
/sas5/ne_interface_types_servlet
/sas5/connection_types_servlet
/sas5/default_ne_parameters_servlet
/sas5/navigation_servlet/showEmptyMaintenance
/sas5/system_status_servlet
/sas5/navigation_servlet/showEmptyOperationalConfiguration
```

```
/sas5/grc_servlet
/sas5/grc_servlet/showParameters?idFFF
/sas5/grc_servlet/showParameters?idFFF
/sas5/grc_servlet/showParameters?idFFF
/sas5/grc_servlet
/sas5/grc_servlet/showParameters?idFFF
/sas5/grc_servlet/showParameters?idFFF
/sas5/ui_user_servlet/logout
```