LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN
Lehr- und Forschungseinheit Medieninformatik
Prof. Dr. Heinrich Hußmann

Projektarbeit

“Usability-Proxy” für Webseiten

Monika Wnuk
wnukm@informatik.uni-muenchen.de

Bearbeitungszeitraum:	1. 3. 2005 bis 31. 8. 2005
Betreuer:	Richard Atterer
Verantw. Hochschullehrer:	Prof. Hussmann

Zusammenfassung

Bei der Usability-Analyse von Websites sind herkömmliche User Tests, oft in Laborumgebungen durchgeführt, aufwändig und teuer. Der “Usability-Proxy”-, kurz “UsaProx”-Ansatz, bietet die Möglichkeit, solche Tests teilweise zu automatisieren und dabei viele der Nachteile von herkömmlichen Usability-Tests und weiteren semi-automatisierten client- und serverseitigen Testprogrammen zu bewältigen. Hierbei werden die Aktionen eines Nutzers, wie Klicks, Mausbewegungen, Tastendrucke, oder Scrollen auf einer Webseite durch den Webbrowser überwacht, der sämtliche angesammelten Daten an den UsaProx-HTTP-Proxy übermittelt. Diese dort in einer Logdatei gespeicherten Daten können weiterführend mittels vielfältiger Auswertungstechniken dargestellt werden. Das Dokument diskutiert das Design und die Architektur des UsaProx-Systems und geht auf die Erweiterkeit bezüglich neuer Arten der Ergebnisvisualisierung ein.

Abstract

During the analysis phase of a website’s usability common user tests are usually being held in laboratory environment and are therefore rather complex and expensive. The “UsaProx”, short for “Usability Proxy”, approach provides the possibility to capture usage data in a semi-automated way by realizing logging through an HTTP-proxy. The tool overcomes many of the problems of traditional and automated client-side and server-side testing techniques. With UsaProx user actions such as clicks, mouse movements, keystrokes or scrolling on a Web page are monitored by the Web browser, which transmits all monitored data to the UsaProx proxy server. The data aggregated in a log file can be further visualized by numerous evaluation and presentation techniques. This document discusses the design and architecture of UsaProx and also describes how it can be extended for new kinds of visualizations.

Aufgabenstellung

Bei der Usability-Analyse von Websites sind User Tests, bei denen Beobachter den Nutzern der Website über die Schulter schauen, aufwändig und teuer. Daher soll in der Projektarbeit ein Programm entwickelt werden, mit dessen Hilfe solche Tests teilweise automatisiert werden können: Die Auswertung der Aktionen eines Nutzers der Website findet durch den Webbrowser statt, der sämtliche Aktivitäten des Nutzers (z.B. Klicks, Maus-Bewegungen, Scrollen auf der Seite, Tastendrücke) an den Server übermittelt. Die Implementierung soll in Form eines HTTP-Proxy erfolgen, sodass keine zusätzliche Software auf dem Rechner des Nutzers installiert werden muss.

Teilaufgaben:

- Implementierung eines einfachen HTTP-Proxy ohne Caching, oder alternativ: Untersuchung bestehender Open-Source-Implementierungen, Einarbeiten in den Code einer Implementierung.
- Erweiterung dieses Proxy: Bei der Auslieferung an den Browser sollen Seiten “on the fly” um JavaScript-Elemente angereichert werden, die ein detailliertes Mitloggen aller Nutzer-Aktionen erlauben
- JavaScript-Programmierung: Erstellen von geeignetem JavaScript-Code, der die Überwachung mindestens von Seitenaufrufen, Klicks, Maus-Bewegungen, Scrollen und Tastendrücken mit sekundengenauen Zeitstempeln erlaubt

- Logging: Aufzeichnung der Rückmeldungen des JavaScript im Proxy, Ausgabe als Logdatei
- Auswertung: Entweder Erstellung eines kleinen Programms, mit dessen Hilfe die Logdatei ausgewertet werden kann oder Demonstration, wie vorhandene Software zur Auswertung benutzt werden kann
- Demonstration der Funktionsfähigkeit an einem kleinen Beispiel

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbstständig angefertigt, alle Zitate als solche kenntlich gemacht sowie alle benutzten Quellen und Hilfsmittel angegeben habe.

München, 30. September 2005

. . . .

Inhaltsverzeichnis

1 Introduction

1.1 Background

Successful user interfaces, even for simple systems, are difficult to create. Particularly, designing interfaces for the Web is challenging, as completing common tasks online often causes inconvenience for users. If a Web site is difficult to use and the website's information is hard to read or doesn't answer users' key questions and if users get lost on a website they leave [7]. Unfortunately, Website usability continues to be a problem. Although many companies now have dedicated usability specialists on staff, and even more companies say that they want to improve the usability of their websites and intranets only an estimated 66 percent of corporate sites provide adequate usability [8]. Therefore, usability testing is necessary to identify and eliminate insuffiencies.

However traditional user testing techniques are inadequate. In general, a group of users, each representative of the target audience, are brought to an usability lab and asked to complete several predefined tasks. Their actions are monitored by an usability engineer or by an eye-tracking tool. Afterwards, the users are interviewed to gain a more detailed insight into the issue.

This type of test not only makes it hard to cover all possible tasks on a site due to the restricted number of participants but also merely leads to representative results. The rather qualitative than quantitative data accumulation makes it difficult to find the majority of usability problems and the lab environment forces the user to act under observation. However tests with large numbers of participants are very time consuming and expensive.

1.2 Objectives

Better tools and techniques are needed to increase the number of participants and to reduce the cost of traditional usability tests. One way to fulfil these requirements is to automate the data capturing and evaluation process. The “UsaProx”, short for “Usability Proxy”, tool introduced in this document was developed for this purpose.

1.3 Structure of the document

The document is divided into the following parts:

- Chapter 2 discusses the requirements for the desired UsaProx tool with regard to the test participants as well as to the usability engineers
- Chapter 3 gives an insight into the design of UsaProx starting with a comparison between related exisiting approaches. Afterwards, the UsaProx approach is described followed by a detailed elaboration of the system architecture and concept
- In chapter 4 the steps and concepts that were necessary to implement the UsaProx system are accurately regarded
- Chapter 5 provides an overview of the types of information that may be identified from the test results and some visualization approaches that may be applied. An exemplary evaluation application is introduced for the illustration.
- Chapter 6 explains which drawbacks occured during system tests and how testing was used to improve the system quality and robustness.
- Finally, chapter 7 summarizes and discusses the UsaProx approach followed by an outlook.

2 Requirements

The quantitative collection of data requires a tool that is accessible to a large number of people without the need to spend time with scheduling and observating them and with analyzing the results afterwards.

Users must have the possibility to evaluate Web sites remotely in their own environments, from any location and at any time using their own equipment and network. Therefore, the tool must work with the standard Web browsers and on every platform without having to be installed on the user's computer. All relevant user actions have to be monitored automatically without attracting the user's attention and not interrupting and burdening him while he is moving through a Web site.

For this purpose, a tool is needed that imperceptibly enhances the HTML-code of the visited Web pages while they are being requested by the user's browser with special browser-independent code that automatically manages the monitoring of the user' actions. The collected data has to be stored in a log file on a central server (at the same location where the tool is located) without slowing up the browser's performance.

For the usability engineers purpose the monitoring and logging of the user's actions must proceed accurately, realtime and reproducable. The monitored data has to be available at any location and any time. It is important to provide a range of features and possibilities that is comparable to those of a standard usability test which results in an accurate listing of what the user actually did while performing the predefined tasks. Therefore, the tool must aggregate the following data:

- Navigation behaviour metrics accross Web sites such as the most frequently used path through a Web site, key entry and exit pages, and page conversions
- Time based metrics such as the average time spent on a page or the visit date and time
- The actions of one single user within a certain Web page such as the page request, mouse clicks, mouse movements, keystrokes, scrolling and window resizing behaviour

Concerning the quantitative aspect of an automatization of the usability testing process, the data collection has to result in an satisfying evaluation and interpretation of the data either in the form of an statistical overview or as a detailed, specific report. Possible visualizations should include:

- General reports with average statistics such as most clicked buttons and links, average number of clicks, average time spend on a page, and visitors, hits and views per page
- Diagrams that illustrate Web pages traversed and paths taken through a Web site
- Mouse movements visualized colored traces on a two-dimensional area
- Overlaying of the user's actions and the Web metrics directly on the currently evaluated Web page or the respective screenshot
- Replaying of user actions on the actual Web pages.

Finally, an examplary evaluation application should be introduced to illustrate some of the possible visualizations applied to the captured data.

3 Design

Previous work has focused on a wide variety of implementations that direct at these requirements.

Benchmark Tools: Static website analysis tools, also called benchmark programs, analyze the HTML code of Web pages to determine if it conforms to a set of usability guidelines. For instance *WatchFire Bobby* [9] and *W3C HTML Validator* [10] provide this functionality. Though many of these tools are used for the evaluation and improvement of the usability of sites and despite of the potential benefits of automated evaluation and transformation tools, it is not evident that they result in sites that are more usable and accessible than those produced or accessed without these tools [1]. Ivory et al. present an empirical study of automated evaluation tools that shows that the tools themselves are difficult to use and, furthermore, suggest that the tools did not improve user performance on information-seeking tasks. Another drawback is that benchmark tools do not involve the user's actions and focus thus do not meet the needs of users. Optimization relates to page contents and coding but not to the view of the user or to more contextual/relative attributes (e.g. target group, purpose of the page).

Tracking Tools (Client-Side Logging): Other approaches include the user's focus into the usability evaluation. Tracking tools record and log the user's actions in an more automated thus more convenient way than traditional eye-tracking methods, which are relatively expensive, difficult to set up and administer, and also limited to the laboratory. For this purpose the user's computer is instrumented with special software or the pages of a Web site that has to be tested are added special code so that usage transactions may be captured.

Projects such as *Lotus' HyperCam* [11] or *ScreenCam* [12] are based on the screen capturing approach. For this purpose, the image data send from the graphic card to the computer display is recorded and copied to a storage device. However the tracking of displayed data doesn't provide any information about human gesticulation or mimic. Therefore, the evaluation of events and actions within a Web page cannot be automated but still requires human assistance.

Likewise Web site analyzer tools such as Hitbox [13] require an adaptation of the Web pages. Special JavaScript functions are pasted into the code that provide real-time web site analysis to show where site visitors come from, what content and products they look at, and which navigational paths they take through the site. Unfortunately, besides the need to modify Web pages Web site analyzers have a similar disadvantage as screen capturers: they do not satisfactory consider the user's behaviour within a Web page.

In contrast the *Enhanced Restricted Focus Viewer (ERFV)* [2] collects data such as mouse clicks along with the path of the user's visual attention automatically as he browses through a Web site. Mouse movements and time stamp data are written to an output file that then can be replayed in the separate Replayeräpplication. One drawback is that the tool works with static files that must be created of each Web page in order to start an experiment.

ObSys [3] works with so-called mousemaps to track and visualize mouse movements and clicks. A mousemap displays the recorded mouse movement traces on a two dimensional area. The mouse trails grow wider in order to signalize a slower movement and change colors to visualize the movement direction. Mouse clicks are also integrated as circles. ObSys also offers the possibility to underlay a screenshot of the evaluated content.

The advantage of the tracking approach is that nearly everything can be transparently recorded and logged, from simple events such as mouse clicks and keystrokes to more complex events such as page requests. Unfortunately, though mainly no other hardware than a personal computer is needed, these tracking tools require software to be installed on the user's computer which test persons may be unwilling or unable to do. This not only may restrict the number of test participants but also limit the usability test participants to experienced users, which may not be representative of the target audience. Futhermore, there needs to be a possibility for sending the recorded data back to the usability testers. Finally, the software is platform dependent and thus may not be compatible with every operating system or browser.

Server Log Analysis: With server logging it is easy to collect large amounts of data of a huge number of users, since nearly every web server automatically records traffic (page requests). Users are able to work remotely from any location without the need to install special software. Log file data reflects actual usage in natural working conditions, compared to the artificial setting of a usability lab. It represents the activity of many users, over a potentially long period of time, compared to a limited number of users for an hour or two each. Furthermore there are many tools for server log analysis available, such as the *Webalizer* tool [14] or *Nihuo Web Log Analyzer* [15]. What server log files provide are details about file requests to a server and the server response to those requests. Collecting and analyzing these files can provide information about who is visiting a Web site, what information they're requesting and their navigation and behavior [4]. However, the approach implies two main disadvantages. The data is for the most part insufficient relating to usability questions. The user's actions on a Web page do not appear in server log files and logging usually occurs temporally delayed and imprecise. Furthermore, server log data is restricted to the owners of a web server. The usability engineer will have to work with the organization's technical staff to make sure the desired data is logged and summarized. The structure of an individual log file varies from server to server and in addition, logging has to be done on every Web server the sites of interest are located on.

What is needed is a method which makes mining log files for finer-grained detail about visitor actions on web pages possible and logs data uniformly on a central location.

3.1 The Proxy Approach

To address these needs, we developed the UsaProx tool that is based upon an proxy approach. With this approach logging is automatically done on an intermediate computer, the so-called proxy, lying between clients, such as a web browser, and Web servers while multiple recruited users surf the Web in order to complete several predefined tasks (see 3.1). The assumption is that all page requests, the browsers make, go through the proxy. After all the data is collected in a log file, the web designer could use special evaluation tools to aggregate and visualize the data to detect usability problems. Once problems are found and fixed, the entire process may be repeated in an iterative design process.

In particular, what is intended to happen when usability is tested with the UsaProx tool is the following:

- The users are fulfilling their predefined tasks visiting the determined Web pages while UsaProx is registered as proxy in the participant's browser properties
- When a user's browser requests UsaProx to forward him a specific Web page, the respective page is fetched from the Web server and prepared with special code before being transmitted to the user

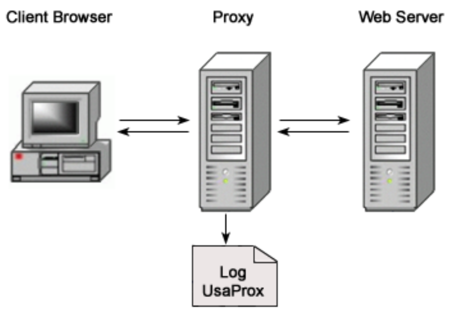


Figure 3.1: **Proxy-based logging is done on an intermediate computer and avoids many of the problems faced by analysis tools that have to be installed on the user's computer and server-side logging**

- During the user's visit on the page all his actions (such as page requests, mouseclicks, mouse movements, scrolling, keystrokes) are monitored and continously transmitted to the UsaProx proxy

- The captured data is stored to a log file

UsaProx may be best and easiest realized as HTTP-proxy. Since most Web requests concern simple HTML-documents and files embedded in those (e.g. images), in nearly most cases the Hypertext Transfer Protocol (HTTP) is used to request and exchange these files. Advantage can be taken of this fact for the interception of both, the HTML files and the log data coming from the client. Regarding the client UsaProx acts as an HTTP server whereas the Web server is transferring files to UsaProx in the form of a Web client. Using the less complex yet adequate HTTP version 1.0 an implementation of the proxy functionality doesn't demand excessive effort but enables the UsaProx tool to act unobtrusive and be compatible with existing operating systems and browsers. In addition, this way of logging is fast and easy to deploy on any Web site.

Furthermore, with common embedded scripting such as JavaScript it is possible to capture nearly every event the user triggers without attracting attention and without any additional software being required on the user's browser or computer. It will also be easier to make it compatible with future operating systems and browsers, such as those found on handheld devices and cellular phones, since scripting techniques will also advance.

With the proxy approach it is further easy to create a log file format that is more useful for usability analysis. The UsaProx log file extends the common server log by information concerning actions and triggered events. Therefore, it is now not only possible to collect large amounts of Web traffic data of a huge number of users but also what these users actually do on the Web pages.

In the rest of this document, the design and structure of the UsaProx system is described.

3.2 UsaProx Functionality

As described above, the UsaProx proxy module is required to implement functionality to process page and log requests and server responses. This functionality represents the behaviour of the system. It may be documented in various ways. The UML notation [16] therefore provides so-called use cases.

Use cases are a means for specifying required usages of a system. Typically, they are used to capture the requirements of a system, that is, what a system is supposed to do. The key concepts associated with use cases are actors, use cases, and the subject. The subject is the system under consideration to which the use cases apply. The users and any other systems that may interact with the subject are represented as actors. Actors always model entities that are outside the system. The required behavior of the subject is specified by one or more use cases, which are defined according to the needs of actors.

For an overview of the required system functionality the UML notation provides use case diagrams. A more detailed view is then formulated in the use case descriptions. Figure 3.2 shows the use case diagram for the UsaProx proxy module.

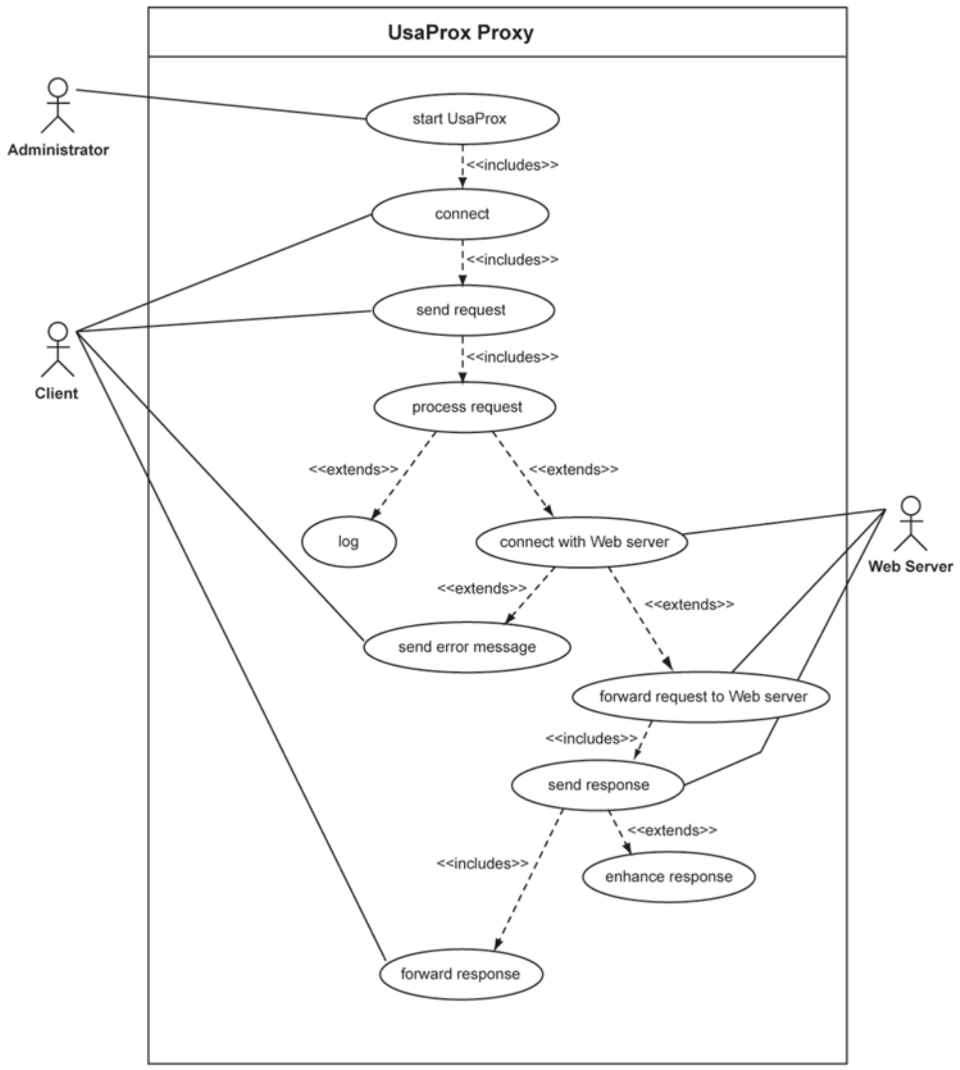


Figure 3.2: **use case diagram of the UsaProx proxy component**

Several use cases build up one main task. For instance, requesting a Web page consists of the use cases “Connect”, “Send request”, “Process request”, “Connect with Web server”, “Forward request to Web server”, “Send response”, “Enhance response”, and “Forward response” whereas logging terminates with the use case “Log” instead of contacting the Web server.

As use case diagrams only represent an overview of the system behaviour and don't make entirely clear what they're about, the following tables give an accurate description of every use case including the scenario it stands for. Tables 12 and 13 describe what is done by the enhancing script block in the user's browser.

No.	**1**
Title	**Start UsaProx**
Description	The administrator starts the UsaProx system, that waits for incoming connection
Precondition	- UsaProx is installed - Java Virtual Machine is installed
Primary Scenario	1. The administrator starts the system 2. UsaProx gets ready for accepting incoming connections
Secondary Scenarios	- System abort through administrator - System may not be started (e.g. another application is already running on the same port, UsaProx port lies between 1 and 1023 without having root permissions on a Unix operating system) - Making ready for accepting incoming connections fails
Postcondition	The system is ready to accept incoming connections

No.	**2**
Title	**Connect**
Description	A client connects to the UsaProx system
Precondition	- UsaProx is running and ready to accept incoming connections - The maximum amount of accepted connections isn't reached yet
Primary Scenario	1. Connection is accepted 2. Input and output channels are created 3. A new Thread object is instantiated for the connection and for handling the client request
Secondary Scenarios	- Connection cannot be created (e.g. maximum amount of connected clients reached) - Connection is aborted by the client (IOException) - New Thread object can not be instantiated
Postcondition	Client is connected with the UsaProx system and the system is ready for accepting the incoming request

No.	**3**
Title	**Send Request**
Description	Client sends the request to UsaProx
Precondition	Client is connected to UsaProx and all communication channels are set up
Primary Scenario	1. Client sends request 2. Request is fetched through input channel
Secondary Scenarios	- Connection is aborted by the client (IOException) - Request not clear
Postcondition	System is ready to process the request

No.	**4**
Title	**Process Request**
Description	UsaProx processes the client request
Precondition	- Client is connected to UsaProx and all communication channels are set up - UsaProx is ready to process the received request
Primary Scenario	1. UsaProx retrieves request type (regular, log, file request) 2a. If regular type, request is transferred to respective Web server 2b. If log type, logging is done 2c. If file type, requested file is transferred to client
Secondary Scenarios	- Connection is aborted by the client (IOException) - Request not clear
Postcondition	System has processed the request

No.	**5**
Title	**Log**
Description	UsaProx logs received data to log file
Precondition	- Client is connected to UsaProx and all communication channels are set up - UsaProx has received a log request
Primary Scenario	1. UsaProx brings received log data into wished format 2. UsaProx connects with log file (creates file output channel) 3. UsaProx writes formatted data to log file 4. The client is send an adequate response message
Secondary Scenarios	- Writing isn't possible (e.g file may not be found)
Postcondition	System has processed log-request

No.	**6**
Title	**Connect with Web server**
Description	UsaProx connects to Web server
Precondition	- Client is connected to UsaProx and all communication channels are set up - UsaProx has received a request
Primary Scenario	1. UsaProx connects to Web server 2. Input and output channels are created 3. A new Thread object is instantiated for the connection and for transmitting the request
Secondary Scenarios	- Connection is aborted by the client (IOException) - Server cannot be connected or doesn't exist; in this case an error message is send to the client
Postcondition	System is connected to Web server and is ready to send request

No.	**7**
Title	**Send error message**
Description	UsaProx sends an error message to client in case of connection errors
Precondition	- Client is connected to UsaProx and all communication channels are set up - An error has occured while connecting to the Web server
Primary Scenario	1. UsaProx generates error message 2. UsaProx sends message to client 2. UsaProx closes all communication channels and connections 3. UsaProx deletes all Thread objects for this request
Secondary Scenarios	- Connection is aborted by the client (IOException)
Postcondition	System has send error message, closed all connections and deleted all related Threads objects

No.	**8**
Title	**Forward request to Web server**
Description	UsaProx forwards the client request to Web server
Precondition	- Client is connected to UsaProx and all communication channels are set up - UsaProx has received a request - UsaProx is connected to Web server and all communication channels are set up
Primary Scenario	1. UsaProx sends request to Web server 2. System gets ready to expect response
Secondary Scenarios	- Connection is aborted by the client (IOException) - Connection is aborted by the Web server (IOException)
Postcondition	System is connected to Web server and is ready to receive response

No.	**9**
Title	**Send response**
Description	UsaProx accepts Web server response
Precondition	- Client is connected to UsaProx and all communication channels are set up - UsaProx has received a request - UsaProx is connected to Web server and all communication channels are set up - The request has been send to Web server
Primary Scenario	1. UsaProx accepts Web server response 2. UsaProx retrieves response content type (html/text, other) 3a. If html/text type, the content-enhance routine is started 3b. All other types of responses are directly forwarded to the client
Secondary Scenarios	- Connection is aborted by the client (IOException) - Connection is aborted by the Web server (IOException)
Postcondition	System has accepted response

No.	**10**
Title	**Enhance response**
Description	UsaProx enhances html/text response with monitoring script
Precondition	- Client is connected to UsaProx and all communication channels are set up - UsaProx has received a request - UsaProx is connected to Web server and all communication channels are set up - Web server response has been accepted - Response type has been determined as html/text
Primary Scenario	1. UsaProx accepts Web server response 2. UsaProx parses the html/text data and finds the insert location 3. The system inserts the script block at the determined location
Secondary Scenarios	- Connection is aborted by the client (IOException) - Insert location cannot be found
Postcondition	Enhanced response is ready to be forwarded to the client

No.	**11**
Title	**Forward response**
Description	UsaProx forwards response (also enhanced response) to client
Precondition	- Client is connected to UsaProx and all communication channels are set up - UsaProx has received a request - UsaProx is connected to Web server and all communication channels are set up - Web server response has been accepted - Response possibly has been enhanced
Primary Scenario	1. UsaProx sends response to client 2. UsaProx closes all communication channels and connections 3. UsaProx deletes all Thread objects for this request
Secondary Scenarios	- Connection is aborted by the client (IOException)
Postcondition	System has send the response, closed all connections and deleted all related Threads object

No.	**12**
Title	**Script: Monitor the user's actions**
Description	The UsaProx script monitors user actions in the browser
Precondition	- JavaScript is activated in the browser properties - UsaProx is registered as proxy in the browser properties - Client has received an enhanced Web page from the UsaProx proxy
Primary Scenario	1. The UsaProx script collects data about page requests, mouseclicks, mouse movements, keystrokes, scrolling 2. The log process is always started after a defined time period
Secondary Scenarios	- Client ends page visit before data could have been collected
Postcondition	An amount of monitored data has been temporarily stored

No.	**13**
Title	**Script: Log monitored data**
Description	The UsaProx script sends the temporarily stored data to the UsaProx proxy
Precondition	- A sufficient amount of data was collected
Primary Scenario	1. The UsaProx script creates log-request 2. The UsaProx script sends log-request to UsaProx proxy
Secondary Scenarios	- Client ends page visit before data could have been send
Postcondition	The collected data has been send to the UsaProx proxy

To further understand the behaviour of the UsaProx system, interactions between the actors and objects in the system while certain use cases are worked out are described. In the UML notation interactions are expressed through behaviour diagrams. In general, behavior diagrams show the dynamic behavior of the objects in a system, including their methods, collaborations, and activities. The dynamic behavior of a system can be described as a series of changes to the system over time.

Behavior diagrams can be further classified into several other kinds. For our needs a sequence diagram, the most common behaviour diagram, shows best what happens between the actors and UsaProx objects during the completion of a task containing several use cases by focusing on the message interchange between a number of lifelines. Figure 3.3 shows the sequence diagram for the task “Requesting an html page” (use cases 2, 3, 4, 6, 8, 9, 10, 11). In figure 3.4 the sequence diagram for the task “Logging” (Use Cases 2, 3, 4, 5) is expressed.

3.3 UsaProx Structure

The term “structure” in this chapter refers to a composition of interconnected run-time instances of objects collaborating to produce the functionality of the UsaProx system.

UsaProx Classes The diagram 3.5 shows a system overview of all classes of the UsaProx System. For every connection/request a ClientRequest class is instantiated that further processes the request. The request type defines which further object is instated by the ClientRequest object to finally complete the requested task.

Classes LogFile and ProxyScript provide the functionality of logging data and sending a requested JavaScript file to the client. At the beginning, both classes are once instantiated by the main UsaProx class. The contained methods log() and sendFile() are then accessed by a ClientRequest class when needed. In this manner an exclusive access to the log and script file can be granted when the log() and sendFile() method are restricted to one operating ClientRequest class at the same time.

In comparison, a ServerRequest object is created directly by the respective ClientRequest object whenever any Web server file is requested (e.g. HTML page, images).

With the above definitions and diagrams all aspects of the UsaProx system were implemented in accordance with the design concept. Read about the implementation of the UsaProx system in the next chapter.

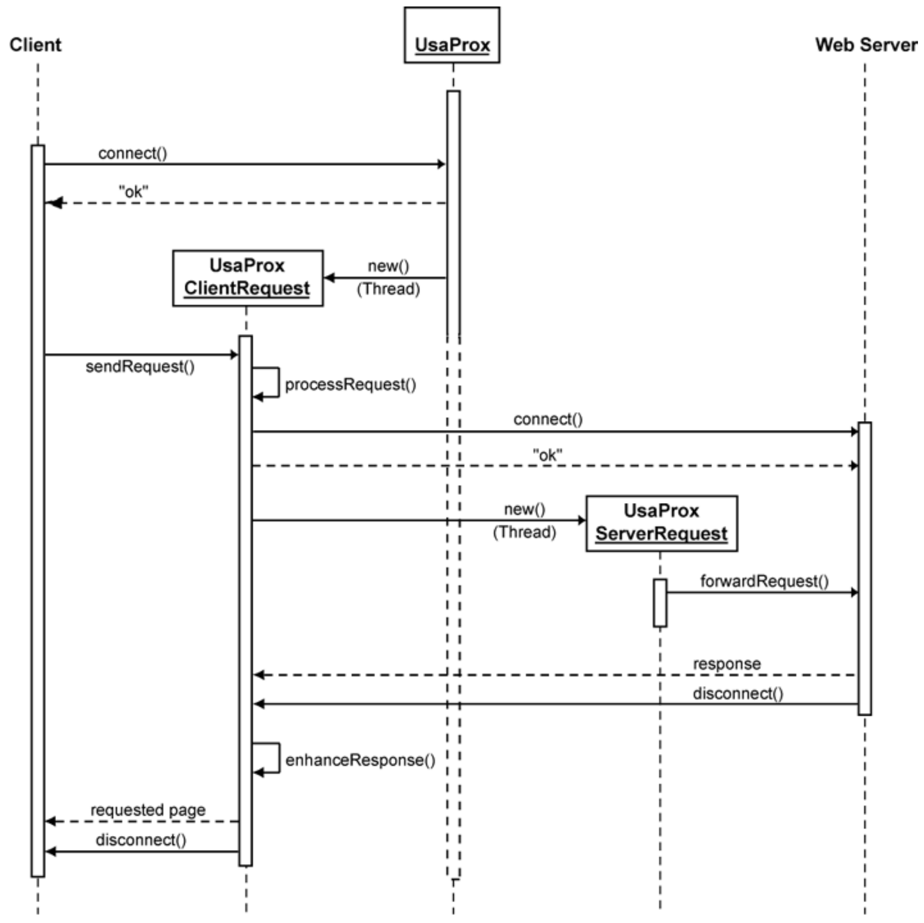


Figure 3.3: **Sequence diagram of the task "Request an html page" consisting of the Use Cases "Connect", "Send request", "Process request", "Connect with Web server", "Forward request to Web server", "Send response", "Enhance response", and "Forward response"**

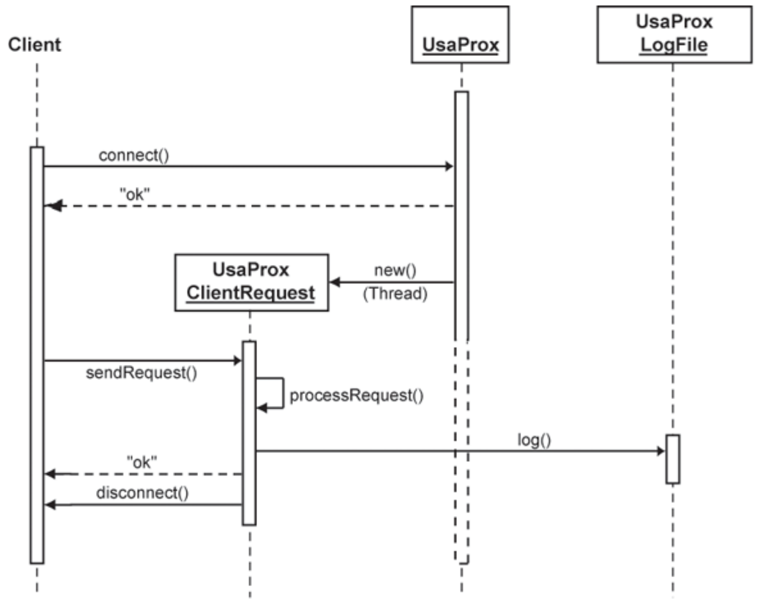


Figure 3.4: **Sequence diagram of the task “Logging” consisting of the use cases “Connect”, “Send request”, “Process request”, and “Log”**

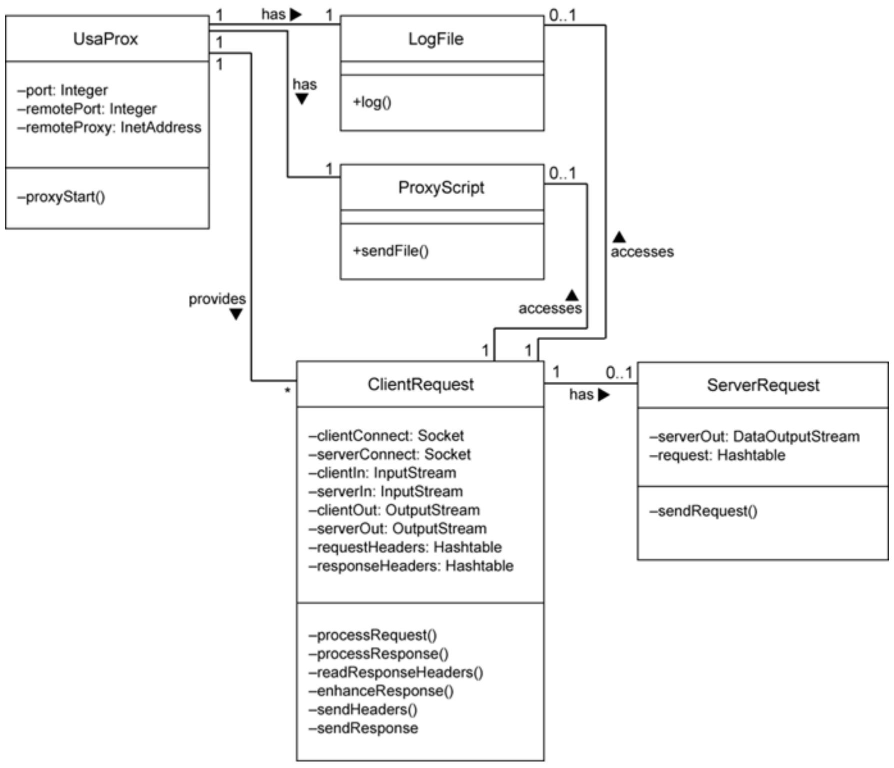


Figure 3.5: **Class Diagram of the UsaProx system**

4 Implementation

The UsaProx system was iteratively realized in 4 steps. In addition, an evaluation program was implemented to better visualize the logged results. The development process of the UsaProx proxy was realized in the following order:

1. Implementation of a simple HTTP-Proxy without caching
2. Enhancement of the proxy with the functionality of adding special script to HTML-pages to have the user's actions monitored in his browser
3. Development of the related monitoring Javascript code
4. Recording of the JavaScript feedback by the proxy and logging it into a log file

4.1 Proxy

The UsaProx basic proxy system is implemented with the object-orientated programming language Java (current SDK version 1.5). Therefore, UsaProx not only is completely platform independent but also highly extensible.

Although many Java implementations of HTTP-proxies are available, for instance Jetty [17], the UsaProx proxy is consciously an independent implementation. Indeed many functions and features of the existing proxies are advantageous when implementing such a project. Those mostly as open source available proxies often provide a wide range of security and contingency avoiding features. Nevertheless, those implementations are generally oriented towards caching Web pages, filtering according to firewall rules, and advanced HTTP-functionalities such as retaining a connection between a client and a Web server over a sequence of requests. They are also oriented towards accepting any kind of file stream from the Web server including encoded streams. Since the UsaProx proxy needs to simply parse ordinary HTML-data and isn't up to filtering, caching nor providing any encoding functions, all that advanced functionality isn't neccessary nor desired. Therefore, the UsaProx proxy is a slimmed-down proxy that provides the functionality to quickly forward non-HTML data and enhance HTML-data by a special function.

4.1.1 Sockets and Threads

UsaProx is running using Sockets. A socket is a form of inter-process communication used to form one end of a bi-directional communication link between two applications over the Internet. A socket on a certain host is defined as the combination of an IP address, a protocol, and a port number. Each socket gets bound to a given port, which lets the transport layer protocol (typically UDP or TCP) identify which application to send the data to.

The basic life-cycle of the proxy according to the Socket concept is defined as follows:

1. A ServerSocket object is created on a particular port specified as argument with the start of the UsaProx system. with the ServerSocket UsaProx is able to act as a server.
2. The ServerSocket listens for incoming connection attempts on that port (that's why this port must be defined in the client browser's properties)
3. When a client attempts to make a connection, a Socket object is generated connecting the client and the UsaProx proxy.

4. UsaProx gets the Socket's input and output streams that communicate with the client
5. UsaProx and the client interact according to the HTTP protocol until it is time to close the connection
6. UsaProx, the client, or both close the connection
7. UsaProx returns to step two and waits for the next connection

A ServerSocket object generally operates in a loop that repeatedly accepts connections. The operating system stores incoming connection requests addressed to a particular port in a queue. After the queue fills to capacity of normally 50 with unprocessed connections, the host refuses additional connections on that port until slots in the queue open up. Many clients will try to make a connection multiple times if their initial attempt is refused.

Since client requests generally are quite complex and thus cannot be processed immediately UsaProx spawns a thread (the ClientRequest object) for every request. In this manner the loop isn't blocked during the handling of a request. Another thread is instantiated for sending the request further to the Web Server, namely the ServerRequest object. Using two threads allows UsaProx to handle input and output simultaneously: it can be sending a request to the Web server while waiting for the server to respond. This is convenient because different clients and servers talk in unpredictable ways. UsaProx must be flexible to handle unexpected situations.

4.1.2 HTTP Communication

As mentioned in chapter 3.1 the proxy communicates with clients and Web servers over the request/response protocol HTTP. The used version is the simpler 1.0 version. With the protocol an HTTP client typically initiates a request by establishing a connection to a particular port on a remote host (port 80 by default). An HTTP server listening on that port waits for the client to send a request. As stated in the RFC 2068 HTTP [18] an HTTP request is composed of the main request string, such as "GET / HTTP/1.0" (which would request the default page of the Web server) followed by an email-like MIME message which has a number of informational header strings that describe aspects of the request. An exemplary HTTP request is displayed in figure 4.1. The request string begins with a method token, such as "GET" or "POST", followed by the Request-URL (in case of a direct connection to the server, the file-path is sufficient) and the protocol version. Some of the headers are optional, while others (such as "Host") are required by the HTTP/1.0 protocol.

```
GET http://www.google.de/index.html HTTP/1.1
Host: www.google.de
Accept-Charset: utf-8,iso-8859-1
Accept-Language: de
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 5.0; Windows NT)
Connection: close
```

Figure 4.1: **A typical HTTP request begins with the request string followed by a set of compulsory and optional headers.**

Upon receiving the request string (and message, if any), the server sends back a response string, such as "HTTP/1.1 200 OK", and a message with compulsory and optional headers of its own, the body of which is perhaps the requested file, an error message, or some other information. An exemplary HTTP response message is displayed in figure 4.2. The first line of a Response message is the Status Line, consisting of the protocol version followed by a numeric status code and its associated textual phrase. In the example line above "200 OK" stands for a successfully received, understood, and accepted action. "404 Not Found" is another possibility in case a problem occured.

```
HTTP/1.1 200 OK
Date: Mon, 23 May 2005 22:38:34 GMT
Server: Apache/1.3.27 (Unix)  (Red-Hat/Linux)
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
Etag: "3f80f-1b6-3e1cb03b"
Accept-Ranges: bytes
Content-Length: 438
Connection: close
Content-Type: text/html; charset=UTF-8
Content-Encoding: gzip
```

Figure 4.2: **A typical HTTP response begins with the response string followed by a set of compulsory and optional headers.**

With the UsaProx proxy in between, a client connects first to the UsaProx application to the specified port. After having received the HTTP request message UsaProx connects to the Web server specified in the request line and with the "Host" header and forwards the request. Before that happens, some modifications must be done to the request string. Since the client doesn't connect directly to the Web server, the request string contains a complete URL. The proxy must take care of retrieving the Web server name and eliminate it from the request string since the Web server only needs to know the file that must be transmitted. Furthermore, the HTTP version number has to be changed to 1.0. The modified "GET"-line in the previous example would look like as follows (see figure 4.3):

```
GET /index.html HTTP/1.0
```

Figure 4.3: **UsaProx produces a modified request string by eliminating the host name form the URL and changing the HTTP version to 1.0.**

Since the RFC 2068 postulates that all traffic routed over a proxy must be distinguishably marked, UsaProx adds or extends the response "Via" header with the current host name and UsaProx version. Figure 4.4 shows a possible Via allocation.

```
Via: computer1 (UsaProx 1.0)
```

Figure 4.4: **Via header with the current host name and UsaProx version.**

If any errors occur during the completion of a request, for instance, the browser is meanwhile closed by the user ("IOException") or the requested host may not be reached, the proxy handles the error as follows: in the first case, the server mustn't be extra notified, but the proxy must close manually the connection. In the second case the client needs to be notified first with a "404 Not Found" message including a short HTML document that mentions both the numeric code and the string. Afterwards the connection may be closed.

4.1.3 Combination with Another Proxy

The proxy provides the possibility to run within a network with another proxy serving as Internet gateway (see figure 4.5). For this purpose UsaProx may be started with the remote proxy's IP and port as arguments. All requests then will be routed to the remote proxy which will be responsible for contacting the Web server on his part.

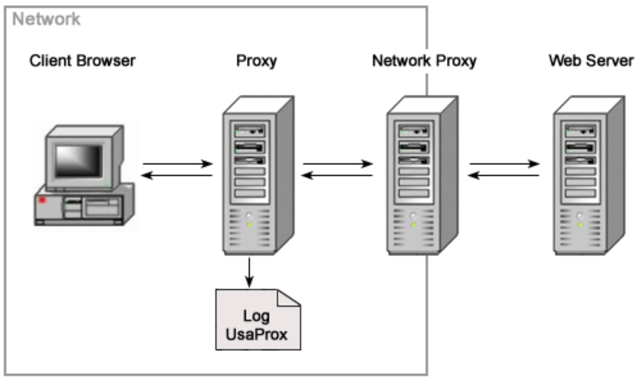


Figure 4.5: **UsaProx proxy model with a network gateway proxy**

4.2 Enhancing Response with Javascript

To have the browser monitor the user's actions, the requested Web pages must be enhanced "on-the-fly" by the UsaProx proxy with special JavaScript code that will manage the actions capturing.

4.2.1 Enhancing Code

The easiest way to add a script to an HTML-page is to only add a reference to a script file containing the actual code located on a virtual Web server. Referencing with JavaScript is done by specifying the "src" attribute of the <script> tag. The code part that is parsed into a Web page by the UsaProx proxy is shown in figure 4.6.

```
<script src='http://lo.lo/proxyscript.js' type='text/javascript'></script>
```

Figure 4.6: **Enhancing <script>-tag with a reference to the actual file located on a virtual host.**

The browser will then request the JavaScript file at the virtual host, in our case "lo.lo". UsaProx will capture the request signified by the very special host name instead of forwarding it in order to send the client the file by himself.

For this purpose a special Java class named SendFile is introduced. This class contains the synchronized function sendFile() that is responsible for the file fetching and sending procedure. Synchronizing functions is based on the assumption that multiple concurrent threads try to access one ressource at the same time. To avoid conflicts the method declaration for the access method of a ressource object contains the synchronized keyword. Whenever a thread enters a synchronized method, it locks the object whose method has been called. Other threads cannot call a synchronized method on the same object until the object is unlocked. Therefore, only one ClientRequest thread may be accessing the JavaScript file in order to send it to the respective client.

Before the insert location for the <script>-block can be searched, UsaProx must check an HTML-coded response is being processed or if merely an image, video/audio, application or other file was requested. For this purpose, the “Content-type” header has to be analyzed. If a text/html type is detected, the enhance procedure may be started. Otherwise the data is simply forwarded.

Script tags generally may be inserted into any section of an HTML document but it is common to have them pasted into the head-section. In the case of text/html data the enhance routine loops through the received bytes and compares them with the byte sequence build from the tokenized <head>-tag. All bytes until <head> are collected and temporarily stored together with the script block before all parts are joined and send out to the client. All following bytes can be ordinarily forwarded.

4.2.2 Header Modifications

Besides the response Via header (see chapter 4.1.2) several other headers must be adapted in case of transferring an HTML-document.

First of all, for an comfortable parsing of the byte stream an encoded stream should be avoided just before the request is transmitted. In general, many servers tend to compress data by an encoding format produced by the file compression program “gzip” (GNU zip) as described in RFC 1952 [19]. To suppress gzip UsaProx modifies the request header “Accept-encoding” by the value “identity”. If a Web server receives a request marked with that value, data will be send non-encoded.

Secondly, when increasing the number of bytes of a document by inserting data the response header “Content-length” must be also increased by the number of added bytes. Browsers assume that the data length corresponds to the number specified by the “Content-length” header. If that number differs from the actual length the code will be cropped. That will lead to an incorrect display. Therefore, UsaProx usually adds the number of bytes representing the script line to the given content length.

4.3 JavaScript for Monitoring of User Actions

Event monitoring in the user’s browser is handled by the client-side scripting programming language JavaScript. JavaScript is most commonly used to create dynamic web pages to improve the design, validate forms, detect the visitor’s browser, create/use cookies, and much more.

In the first instance JavaScript is meant to change something on the page. For this purpose, it needs access to the HTML document, for instance to check user input in a form or to change an image src onmouseover. Therefore all browsers have a Document Object Model (DOM) [20], which gives access to the elements of HTML documents. One major use of web-based JavaScript is to write functions that are embedded in or included from HTML pages and interact with the DOM of the page to perform tasks not possible in static HTML alone, such as opening a new window, checking input values, changing images as the mouse cursor moves over, or, in our case, just to capture events.

4.3.1 Events

Events are actions that can be detected by JavaScript. Every element on a web page has certain events which can trigger JavaScript functions. For example, the onClick event of a button element can be used to indicate that a function will run when a user clicks on the button. OnLoad is another event that is triggered when a Web page is loaded. In fact there are over 50 events in all that can be divided into mouse, key and more conceptual events such as onLoad or onFocus. For our purpose the following events are interesting:

- onClick: User clicks on an DOM element
- onMousemove: Mouse is moved over an element
- onKeyPress: User presses a key on his keyboard
- onLoad: A page is loaded
- onFocus: An element is focussed
- onBlur: An element loses focus
- onUnload: A document is unloaded
- onResize: A browser window is resized

Some actions will fire two events. Moving the mouse into or out of an element will fire both a mousemove event and a mouseover (or mouseout) event thus only recording the first one is sufficent.

4.3.2 Registering Event Handlers

To make sure that the browser executes some code whenever the respective event takes place an event handler is registered. The event handler waits until a certain event, for instance a click on a link, takes place. When it happens it handles the event by executing the code preferably defined in a function.

There are four models for registering event handlers: inline, traditional, W3C and Microsoft. Most often event handlers are registered directly within the element node (inline model). For instance an onClick event handler may be defined within an <input>-tag as shown in figure 4.7

```
<input type="button" onClick="someFunction()">
```

Figure 4.7: **Event handler registration with the inline model**

The inline model is not applicable for the UsaProx application since the JavaScript code is defined in an external file. With the traditional model it is possible to define event handlers directly within the JavaScript block. Most browsers, and in fact all Microsoft and Netscape browsers version 4 and higher, accept the code shown in figure 4.8 as a correct way to register an event handler. Whenever the user clicks on the HTML element, the function someFunction() is executed.

A distinct drawback of the traditional model is that the onclick property can contain only one function. This becomes a problem when an event handler shall be attached to multiple already registered ones for the same event. It cannot be guaranteed that no already registered event handlers

```
element.onclick = someFunction;
```

Figure 4.8: **Event handler registration with the traditional model**

are overwritten by the UsaProx script. Likewise our script also may be removed by JavaScripts that are placed below the UsaProx script tag.

To address this problem two advanced event registration models were developed: W3C's and Microsoft's [21]. The W3C model works in Netscape 6 and Konqueror/Safari, the Microsoft model in Explorer 5+ on Windows. Both models work in Opera 7. With the new models overwriting is ommitted by simply attaching functions to the existing ones of the already used event handlers. With the W3C model events are attached using the addEventListener() method. To register an event with Microsoft's model the attachEvent() method is called.

Unfortunately, neither registration model is cross-browser supported and as mentioned above both expose different objects and methods to the script. It is therefore necessary to do a bit of object and method detection. First it has to be checked if the method (such as addEventListener()) actually exists and therefore is familiar to the current browser, and if it does it is used. By having such a distinction between the different browser related methods both models may be combined to achive browser independency. With that combination attaching an function that should be evoked when a click event occurs would look as follows (see figure 4.9:

```
if(attachEvent) {     // Microsoft
    element.attachEvent('onclick', someFunction)
}
else {     // Netscape
    element.addEventListener('onclick', someFunction, false)
}
```

Figure 4.9: **Attaching an event listener either with Microsoft's or with W3C's advanced event registration model.**

4.3.3 Event recording

With the advanced registration models it is now possible to attach the UsaProx specific monitoring functions without influencing the present ones. Since the UsaProx script isn't aware of what elements are used in the document and what names or ids have been assigned the functions must be defined for the entire document resp. window. Key and mouse related event handlers are assigned to the document element whereas the overall load, focus, blur, and unload events are captured by the window.

Events are objects with properties. The properties of interest for the usability analysis are for instance the source element of a click event or the current mouse position. Events triggered on a certain DOM level are usually forwarded to the root element. Any click on a button or link will "bubble" to the document element. So every lower-level event will be recognized and the respective event properties will be available. The following event properties are of interest for the recording:

- onClick: the source element (Netscape: “target”, Microsoft: “srcElement”) and the x and y offset of the mouse position from the top left corner of the window (Netscape: pageX, Microsoft: “offsetX”)
- onKeypress: the key that was pressed (Netscape: “which”, Microsoft: “keyCode”)
- onMousemove: the x and y offset of the mouse position
- onLoad: the width and height of the browser window (Netscape: “window.innerWidth”, Microsoft: “document.body.offsetWidth”, height analog)
- onResize: the new width and height of the browser window

The scrolling event cannot be recorded always when the onScroll event is triggered because it will be triggered with every sligth touch of the scroll bar. That would result in a huge number of monitored scroll events. In contrast an one-time information about a scroll operation is needed. JavaScript therefore offers the definition of intervally released functions. The UsaProx script calls a scroll check function every second as is initiated at the beginning (see figure 4.10). This function tests if the position of the scrollbar has changed over the last interval. For this purpose, Netscape provides the window’s pageYOffset property while Microsoft offers the document.body.scrollTop attribute.

```
scrollCheck_UsaProx = window.setInterval("processScroll_UsaProx()",1000);
```

Figure 4.10: **The scroll event check is initialized to be processed once a second.**

The recording of mouse movements is based on a similar approach. Since the mouse is always moved the logging would blast the log file capacity. Therefore the mouse position is saved once an interval of definable duration.

Every event is recorded with a precise timestamp of the form “2004-12-31,23:59:59”, the currently visited page, and the event with the respective properties.

4.3.4 Transmitting the Recorded Data

Since JavaScript’s influence is restricted to the browser window and the current document it doesn’t provide any direct access to a Web server. Therefore the recorded data usually cannot be send to the UsaProx proxy. Logging is managed in another way: an Image object is instantiated and it’s source is set to the already in a similar way occured virtual URL “http://lo.lo/img.jpg” with the recorded data string as parameter. An image file request is triggered automatically. The instructions are as follows (figure 4.11):

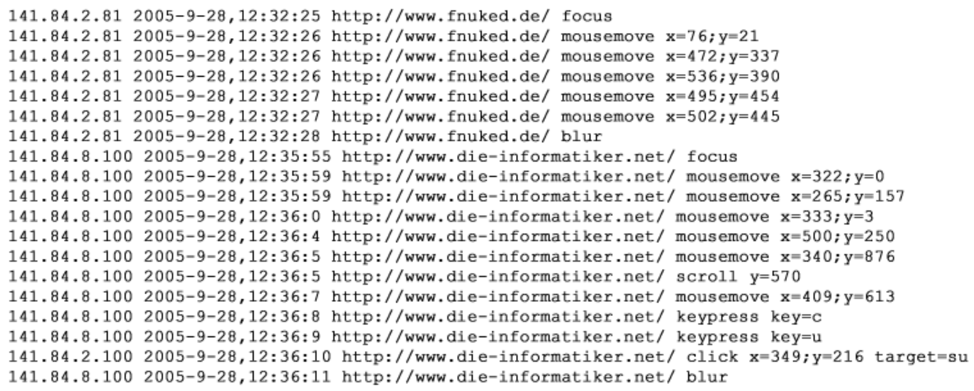
```
var loadImage = new Image();
loadImage.src = "http://lo.lo/img.jpg?" + logData_UsaProx;
```

Figure 4.11: **A tricky method to send data to the UsaProx proxy is to attach it to an image request.**

4.4 Logging

UsaProx captures the log request similar to the script file request (see chapter 4.2.1). While processing the request message the proxy detects the virtual URL and initiates a log request. The log() method is located in another special Java class also intented to synchronize access of concurrent ClientRequest threads to the log file. Therefore, this method's declaration also contains the synchronized keyword. Actually it is more important to grant exclusive access to the log file than to the requested JavaScript file because in case of the log file data is written whereas the script file is only fetched. By synchronizing write access lost updates may be avoided.

What is done by the log() method first is to retrieve the client's host address (IP), since every client may be identified over its socket. The parameter string with the recorded data burst is attached and all is written to the log file. Figure 4.12 shows an extract from the UsaProx log.



```
141.84.2.81 2005-9-28,12:32:25 http://www.fnuked.de/ focus
141.84.2.81 2005-9-28,12:32:26 http://www.fnuked.de/ mousemove x=76;y=21
141.84.2.81 2005-9-28,12:32:26 http://www.fnuked.de/ mousemove x=472;y=337
141.84.2.81 2005-9-28,12:32:26 http://www.fnuked.de/ mousemove x=536;y=390
141.84.2.81 2005-9-28,12:32:27 http://www.fnuked.de/ mousemove x=495;y=454
141.84.2.81 2005-9-28,12:32:27 http://www.fnuked.de/ mousemove x=502;y=445
141.84.2.81 2005-9-28,12:32:28 http://www.fnuked.de/ blur
141.84.8.100 2005-9-28,12:35:55 http://www.die-informatiker.net/ focus
141.84.8.100 2005-9-28,12:35:59 http://www.die-informatiker.net/ mousemove x=322;y=0
141.84.8.100 2005-9-28,12:35:59 http://www.die-informatiker.net/ mousemove x=265;y=157
141.84.8.100 2005-9-28,12:36:0 http://www.die-informatiker.net/ mousemove x=333;y=3
141.84.8.100 2005-9-28,12:36:4 http://www.die-informatiker.net/ mousemove x=500;y=250
141.84.8.100 2005-9-28,12:36:5 http://www.die-informatiker.net/ mousemove x=340;y=876
141.84.8.100 2005-9-28,12:36:5 http://www.die-informatiker.net/ scroll y=570
141.84.8.100 2005-9-28,12:36:7 http://www.die-informatiker.net/ mousemove x=409;y=613
141.84.8.100 2005-9-28,12:36:8 http://www.die-informatiker.net/ keypress key=c
141.84.8.100 2005-9-28,12:36:9 http://www.die-informatiker.net/ keypress key=u
141.84.2.100 2005-9-28,12:36:10 http://www.die-informatiker.net/ click x=349;y=216 target=su
141.84.8.100 2005-9-28,12:36:11 http://www.die-informatiker.net/ blur
```

Figure 4.12: **Extract of the UsaProx log file.**

A simple text format is use for the log file. It resembles the common server log file format as it also contains the client IP and the unified timestamp. However the event part distinguishes the UsaProx log from the common ones and hence makes it useful for the usability analysis.

After having processed the log request, UsaProx must send something back to the client who is expecting a response. One possibility would be to send an empty gif. Unfortunately, browsers store received files and documents in the cache and only request them from the Web server if any modifications are detected. Therefore a "File not Found"-page is generated and send preceded by the 404 status code. In that way the client will request the same image source on and on.

5 Evaluation of the Logged Data

An exhaustive Web site statistics evaluation based on an ordinary server log analysis combined with a more in-depth tracking of what the user actually does when he is exploring a certain Web page provides a various number of possibilities for the evaluation of the results of a task-based usability test. In the next sections the possible kinds of information are explored followed by several ways of visualize them. Finally an exemplary visualization tool is introduced to underline some of the capabilities.

5.1 Identifiable Information

By having automatically allocated the client-side monitoring UsaProx script within every visited Web page various information about user behaviour can be identified from the aggregated data from several test sessions.

Important indicators to look for when analyzing the results of a task-based usability test primary include identifying the overall statistics that are well known from regular Web site analyzations. Since the information about all visitors and an ordered listing of all their Web page visits with an accurate time stamp is aggregated in the log file characteristics such as the paths users take (click through path), the page conversion, and key entry and exit pages can be recognized and classified. Various timebased metrics such as average time spent on a page or the visit date and time can be retrieved. In that way the differences in browsing behavior and real estate “hot spots” may be identified to help prioritize critical content.

Analyzing the browser window size and scrolling behaviour gives a first insight into the user’s preferences to may optimize the Web page content. This feature can be easily extended by let the UsaProx script additionally identify the user’s browser version, his screen resolution, installed plug-ins, the operating system, his available band-width, and so on.

What makes UsaProx extraordinary are the additional event-oriented indicators aggregated in the log file. Besides page entering and window resizing events, clicking, stroking keys, moving the mouse and scrolling give an deeper insight into the user’s intentions and problems they had when accomplishing the given tasks. The time a user spend with exploring a Web page before leaving it and his actions performed in the meantime can be accurately identified.

The tracked mouse movements visualize the user’s focus within a Web page, whether they hesitate on other interesting links/text areas before clicking, and recognize content relating hot spots. This may help to discover either where users encounter obstacles such as confusing navigation, difficult to find links, and missing information or refer to more dominating thus distracting page elements. Furthermore, a non-hesitating mouse movement straight to a link can exhibit a familiarity with the task of finding that link on that page.

Likewise the scrolling of a Web page may signify misunderstandings and orientation problems. Do users scroll or do they evaluate distracting text or image information before they get to the right button? Do users need various attempts to type in data into the right form field? Did they go through the whole information on a Web page before they left? Do they leave the page in-between before they accomplish a task? The conclusions to these questions all may be drawn from the log file, at least when concerning the aggregated results from large amounts of data.

Finally, the most popular links that led to a particular page to see if visitors are navigating the site the way it was intended may be recognized.

5.2 Visualization Capabilities

Potential visualizations range from ordinary listings of Web metrics known from Web site statistics to complex screenshot-underlyed elaborations.

General statistics contain an accurate traffic report often placed in tables that includes listings of all visitors, their page visits (hits and views), visit entry points, the most common navigational paths through a Web site, window width, most common browsers and platforms sometimes accompanied with percentage diagrams. A regular report might easily be extended by usability concerning aspects. Therefore, the detail of a respective Web page could include use data such as the average time spend on the page before a link was clicked, scrolling activites, window resizing, average mouse positions, the most clicked links/buttons, and so on. The drawback of a mainly textual report is obvious: it is hardly imaginable what the users really did and what their focus was while accomplishing the tasks. Therefore it is neccessary to combine statistical information with a more expressive illustration of user behaviour.

Web pages traversed and paths taken may be for instance visualized by diagrams containing Web page thumbnails and connecting arrows. Similar to the WebQuilt visualization component [5] as shown in figure 5.1. Thicker arrows might indicate more heavily traversed paths. The optimal, designer-defined path could be emphasized by a certain color. The Figure illustrates an exemplary thumbnail diagram with a frequently traversed optimal path. This visualization might be combined with a number of symbols representing for instance the time that was spend on the respective page, or buttons that were pushed. Since the UsaProx log contains detailed information about which users visited which pages and what they actually did there along with an accurate time stamp the traversed paths might be easily calculated and illustrated in this manner.

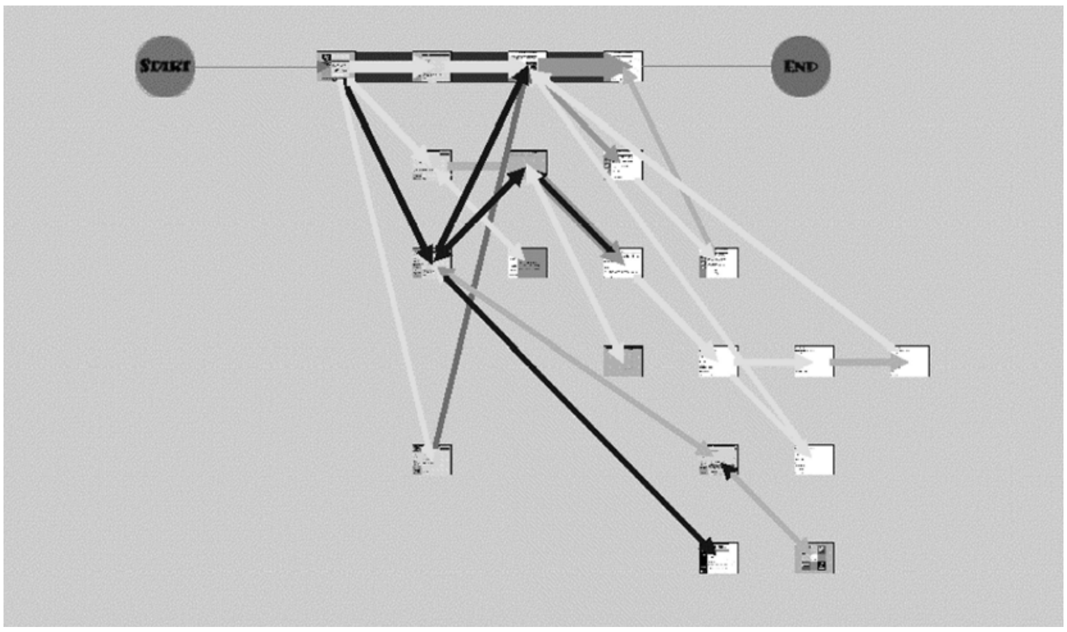


Figure 5.1: **An example WebQuilt visualization. Thicker arrows indicate more heavily traversed paths. The thick black arrows along the top mark a designer indicated optimal path.**

Mouse movements can be visualized by lines ranging from simple, edged constructions to fluid, dynamic paintings as e.g. ObSys mousemaps (see chapter 3) use them. As mentioned above the mouse trails are interpreted according to their speed and direction. The faster a movement was performed the thinner the line is painted. Color gradients symbolize the movement direction. For instance a movement from left to right proceeds from red to green. Figure 5.2 shows an exemplary mousemap. This approach also can be applied to the UsaProx log since the movement velocity and direction may be calculated from the time differences between two movement events and the related mouse positions taking mouse clicks into account.

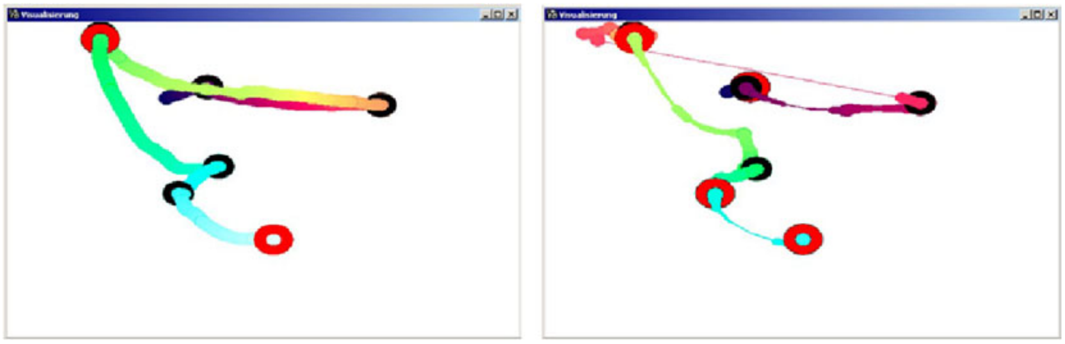


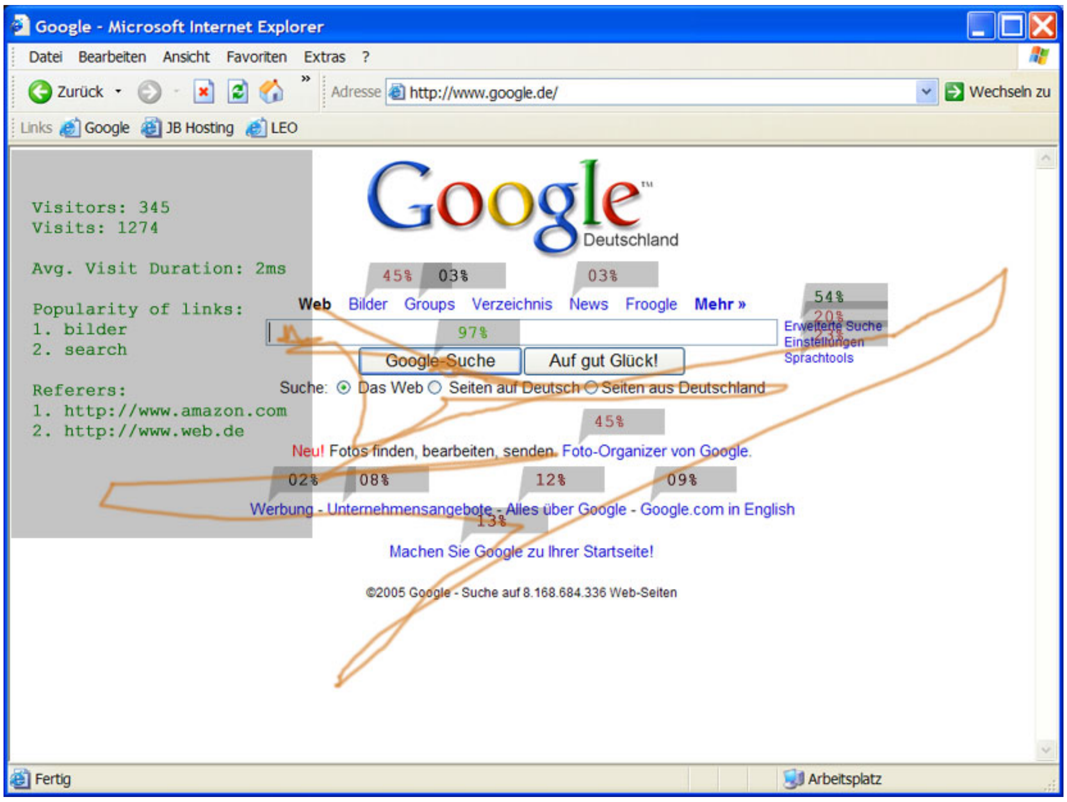
Figure 5.2: **Mousemaps can visualize the speed and direction of mouse movements by varying line thickness and color gradients.**

A more complex evaluation approach would overlay the user's actions and the Web metrics directly on the currently evaluated Web page as drafted in figure 5.3, either in the form of a screenshot or directly within the browser. The popularity of links and buttons might be displayed in a dedicated part of the screen together with other statistic information, for instance the average visit duration or a listing of the pages visitors were at before they arrived at the current Web page. Percentages tagged on each link or button would show results at a glance and color-coding could be used to make it easy to distinguish top-performing links from the less popular ones. Mouse movements would also overlay the screenshot or the real page. Beyond, as extension of this approach a replaying of the actions of a certain user on a certain page could be imaginable.

5.3 Examplary Evaluation Application

An examplary program was implemented in order to visualize some of the possible evaluation techniques. Figure 5.4 shows the grapical user interface seperated into an upper and a lower part. In the upper part a list containing all Web pages that were visited during the usability test is displayed. By clicking on a page in the list the detailed evaluation report is presented below. Visually-left the metric information about the amount of visitors and visits together with the average, minimum and maximun visit duration and number of clicks, and a listing of the most clicked links and buttons is displayed. On the right hand side the mouse movements of all visits distinguished by different colors is visualized. The figure illustrates the evaluation of an exemplary UsaProx usability test on www.google.de. Note that the evaluation program definitely does not cover the whole range of evaluation capabilities that can be done with the UsaProx log since it only intends to demonstrate some of them.

The report data is calculated from the log file contents in the following way: for every visitor distinguished by IP all visits are stored in an index structure. Every visit is represented by an own list structure that enables uncomplicated access to every entry. A visit is defined starting with a focus event and ending with a blur event. In that way the duration of a single visit can be retrieved from the difference between the last and the first entry's time stamp recognized for this



```
Visitors: 345
Visits: 1274

Avg. Visit Duration: 2ms

Popularity of links:
1. bilder
2. search

Referers:
1. http://www.amazon.com
2. http://www.web.de
```

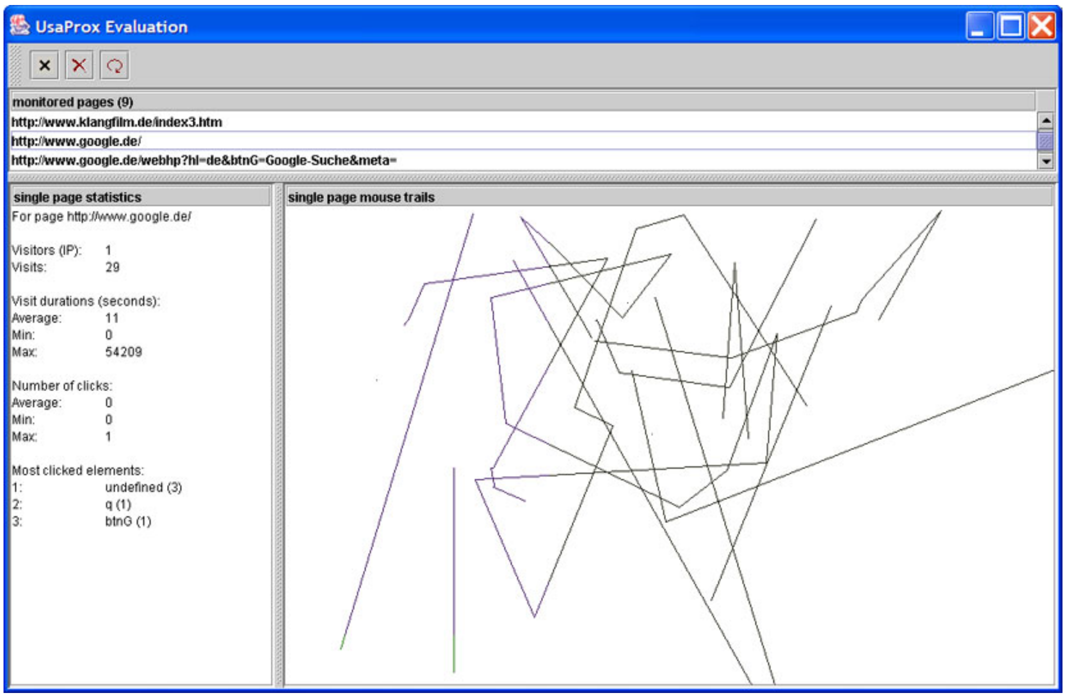
Google - Microsoft Internet Explorer
Datei Bearbeiten Ansicht Favoriten Extras ?
Zurück
Adresse http://www.google.de/
Wechseln zu
Links Google JB Hosting LEO
Google Deutschland
45% 03% 03%
Web Bilder Groups Verzeichnis News Froogle Mehr »
54% 20%
Erweiterte Suche
Einstellungen
Sprachtools
97%
Google-Suche Auf gut Glück!
Suche: Das Web Seiten auf Deutsch Seiten aus Deutschland
45%
Neu! Fotos finden, bearbeiten, senden. Foto-Organizer von Google.
02% 08% 12% 09%
Werbung - Unternehmensangebote - Alles über Google - Google.com in English
13%
Machen Sie Google zu Ihrer Startseite!
©2005 Google - Suche auf 8.168.684.336 Web-Seiten
Fertig
Arbeitsplatz

Figure 5.3: **Evaluation data overlayed on a web page.**

visit. Thereby, the number of clicks also can be easily counted and the most clicked buttons and links can be identified. For every visit an own mouse trail is computed which is composed of line segments whose starting and ending points are retrieved from the mouse position coordinates of two following "mousemove"-event list entries.

The program is entirely implemented with Java and is built with the model-view-controller (MVC) methodology [6] that is illustrated in figure 5.5. The log file data represented by the visitor index structure and the visit list structures and various access methods (getter and setter methods) define the model, the core application. The view renders the model into a form suitable for interaction, the graphical user interface with the displayed usability statistics seperated into the respective areas (views). Additionally, the view contains controllers that listen and respond to events, such as the selection of a Web page in the list or the click on the update-button. Does the user for instance choose a certain Web page from the selection list the following control flow is generated:

- The list controller receives notification of the user action from the selection list object.
- The controller accesses the model, updating and calculating the statistical data concerning the chosen Web page.
- After having updated the Web page data the model notifies all registered "observing" views by sending the update message which newly generates the lower page detail part of the GUI according to the new model data.
- The user interface waits for further user interactions, which begins the cycle anew.



```
http://www.klangfilm.de/index3.htm
http://www.google.de/
http://www.google.de/webhp?hl=de&btnG=Google-Suche&meta=

For page http://www.google.de/

Visitors (IP):      1
Visits:             29

Visit durations (seconds):
Average:            11
Min:                0
Max:                54209

Number of clicks:
Average:            0
Min:                0
Max:                1

Most clicked elements:
1:                  undefined (3)
2:                  q (1)
3:                  btnG (1)
```

Figure 5.4: **GUI of the exemplary UsaProx evaluation program that shows the report of an usability test on www.google.de.**

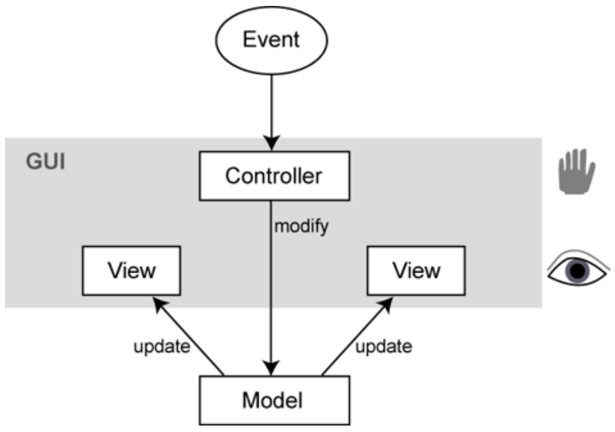


Figure 5.5: **Outline of the control flow in an MVC application.**

6 Usage and Test

During the testing of the UsaProx system several intricacies and drawbacks were detected and eliminated which resulted in a improvement of the stability of the implementation.

Tests have shown that only valid headers conforming to the HTTP protocol are accepted by clients. Response messages that do not seperate the header section from the data through a line wrap weren't correctly interpreted. Headers were displayed together with the HTML code as ordinary textual page content. Leaving a blank line between headers and content resulted in a correct processing within the browser. Although tests proved that headers are case-insensitive and are accepted by browsers in an arbitrary order, we decided to maintain their order to conform to the standard.

Tests were performed with different types of Web pages and sites including sites being hosted on Web servers that tend to send data in the zipped format gzip and those without. Concerning those that encode streams, for instance google.de, web.de, and amazon.com, a parse error was detected at UsaProx while trying to find the head-tag in an apparent html document. Thus the UsaProx script couldn't be inserted and monitoring failed. After having set the request header accept-encoding to the value "identity", the Web servers stopped encoding contents and sent the untreated code.

Concering the content-length attribute of the HTTP response message tests were performed without adapting this field. That resulted in a cropped display of the Web pages. The most bottom part of the code, actually of the script string's byte length, was missing. Increasing the content-length value brought the correct results.

UsaProx was also tested with the different request methods that occur, most frequently "GET" and "POST". UsaProx is indeed working with both main browsers, the Netscape Mozilla from version 6.0 and the Microsoft Explorer from version 5.0 due to the advanced event handler registration model. Since those and higher versions are the most usual ones and users can still be picked out, that's sufficient. UsaProx runs on Unix and Microsoft platforms since Java and in particular the Java.net components used for the proxy are platform independent and portable. Same with the monitoring script: Javascript may be practiced nearly everywhere.

UsaProx barely impairs the response time of requests and the performance of Web pages. It was tested with a couple of users evaluating Web pages over the proxy. The test persons confirmed that the build up of a Web page seemed to need a similar amount of time as without the proxy in-between and the visit wasn't disturbed by any logging approaches except by the switching notifications in the status line which can be disabled.

Most important was to find out if the accumulated data would contain the desired values and granularity. With a lot of tests the log could be improved and adapted to the needs of the users. Above all the mouse movement tracking could be brought to an acceptable grade of detail. At first the recording period was set to 50 milliseconds. That drove to the already mentioned flooding of the logfile. Then a much greater number of 1000 milliseconds was used. That interval proved to be to long. Mouse movements could only be interpreted as rough lines. An amount of 300 milliseconds exposed to be accurate enough but not to close to the last recording. Figure 6.1 shows on the left hand side a too accurate thus log-exploding mouse trail logging while the right screen exposes a too sparse logging interval.

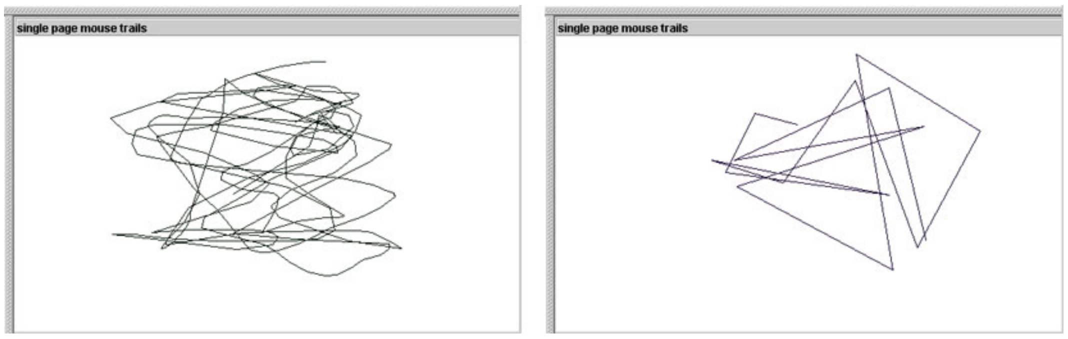


Figure 6.1: **Left screen: Too accurate logging. Right screen: Too sparse logging.**

7 Conclusion

UsaProx is an extensible tool for a semi-automated web usage capturing process. It combines the advantages of client-side tracking software with those of server-side logging in a proxy-based approach that overcomes many of the problems encountered with those. UsaProx may be quickly and easily deployed on any platform and used for any Web site and with a wide range of browsers. Users can perform predefined tasks in their familiar environment without being affected or disturbed. It can be combinated with other usability tools such as online surveys and users can be invited electronically.

The quantitatively aggregated data of a large number of test participants can be evaluated by many possible visualization techniques. Since the introduced exemplary evaluation program only shows some of the analysis capabilities, further tools can be easily built and applied to the UsaProx log. The UsaProx tool also can be extended with various functionalities both client-side and server-side.

The UsaProx approach is a first step to automate the usability testing process. Although UsaProx does not replace the need for inventing tasks participants have to complete and the inviting of test participants, the tool provides a framework that can also be easily extended and applied to other research and optimization fields such as for a semi-automated testing of Web accessibility or the fun of use of Web sites.

Inhalt der beigelegten CD

- Projektarbeit: die eigentliche Ausarbeitung
  - projektarbeit.pdf: Elektronische Version der Arbeit im pdf-Format
  - Projektarbeit Source: Elektronische Version der Arbeit im Originalformat LaTax inkl. Bilder
  - Referenzen: Elektronisch verfügbare Referenzen im pdf- und txt-Format
  - Zwischenbericht: Präsentation im ppt-Format
- UsaProx 1.0: die ausführbaren Programmdateien
  - UsaProx1.0.jar: der eigentliche Proxy im ausführbaren jar-Format
  - UsaProxEvaluation.jar: das beispielhafte Auswertungsprogramm im ausführbaren jar-Format
- UsaProx 1.0 source: die Originaldateien im java-Format
  - UsaProx1.0 src.jar: die original Proxy-Dateien
  - UsaProxEvaluation src.jar: die original Auswertungsprogramm-Dateien
- readme.txt: Installations- bzw. Starthinweise
- inhalt.txt: Inhaltsverzeichnis der CD

Literatur

[1] M. L. Ivory, M. Y.: Using Automated Tools to Improve Web Site Usage by Users with Diverse Abilities. IT and Society, Volume 1, Issue 3, 2003, pp. 195-236

[2] Tarasewich, P. Fillion, S.: Discount Eye Tracking: The Enhanced Restricted Focus Viewer. College of Computer and Information Science, Northeastern University Boston, 2005

[3] Gellner, M.: Mousemaps – ein Ansatz für eine Technik zur Visualisierung der Nutzung von Software und zur Automation der Entdeckung von Bedienungsfehlern. Mensch und Computer 2003: Interaktion in Bewegung, Stuttgart, 2003, p. 197-206

[4] Tec-Ed, Inc.: Assessing Web Site Usability from Server Log Files. White Paper, 1999

[5] H. H. W. Landay, J. A.: WebQuilt: A Proxy-based Approach to Remote Web Usability Testing. University of California at Berkeley, 2001

[6] Bergin, J.: Outline of Model-View-Controller paradigm as expressed in the Java libraries. 1996-1999

Web-Referenzen

[7] J. Nielsen: Alertbox: Usability 101: Introduction to Usability `http://useit.com/alertbox/`, accessed March 12, 2005.

[8] J. Nielsen: Alertbox: Ten Years `http://useit.com/alertbox/`, accessed August 24, 2005.

[9] WatchFire Bobby Worldwide `http://bobby.watchfire.com/bobby/html/en/index.jsp`, accessed March 22, 2005

[10] World Wide Web Consortium: W3C HTML Validation Service `http://validator.w3.org/`, accessed March 22, 2005

[11] Hyperionics HyperCam `http://www.hyperionics.com/hc/index.asp`, accessed March 22, 2005

[12] Hyperionics HyperCam `http://www.hyperionics.com/hc/index.asp`, accessed March 22, 2005

[13] WebSideStory Hitbox Professional `http://www.websidestory.com/products/web-analytics/hitbox-professional/`, accessed March 22, 2005

[14] The Webalizer `http://www.mrunix.net/webalizer/`, accessed March 22, 2005

[15] Nihuo Web Log Analyzer `http://www.loganalyzer.net/`, accessed March 22, 2005

[16] Unified Modeling Language (UML) `http://www.uml.org/`, accessed April 04, 2005

[17] Jetty Java HTTP Servlet Server `http://jetty.mortbay.org/jetty/`, accessed March 21, 2005.

[18] RFC 2068: Hypertext Transfer Protocol `http://www.w3.org/Protocols/rfc2068/rfc2068.txt`, accessed April 25, 2005.

[19] RFC 2068: GZIP File Format `www.gzip.org/zlib/rfc1952.pdf`, accessed July 27, 2005.

[20] Document Object Model (DOM) `http://www.w3.org/DOM/`, accessed June 14, 2005.

[21] P. Koch: Advanced event registration models `http://www.quirksmode.org/js/events_advanced.html`, accessed June 14, 2005.