Project Work

User Clustering and Load Testing

Student:	Sergei Pavlov
Matriculation Number:	31379
Degree course:	Information and Media Technologies
Supervising examiner:	Prof. Dr. Joachim W. Schmidt
Supervisor:	Dr. Hans-Werner Sehring
Delivery day:	6th April 2006

Table of Contents

1. Introduction

This chapter describes the motivation, goals and structure of the project work.

1.1 Motivation

A large number of software projects are designed to serve a large number of concurrent users. Most often these software systems have a web-based presentation layer, quite complex structure, and a significant amount of dynamically generated content. An example for this would be any web portal. They are also expected to meet the non-functional requirement of being capable of coping with a certain number of concurrent users. While the performance of such software might depend on many factors, it is possible to ensure that the system can meet this requirement only by launching it and observing how it works, or testing it as close as possible to the "real-life" conditions. If the software project has not yet been launched or it is not too popular, predicting users' behavior is difficult and imprecise. Otherwise, the behavior can be derived from the system's logs. If the system is already in use and its performance is to be improved, the load generated by users is to be simulated on test server(s) and the system would need to be tuned therein. This study concentrates on improving the performance of existing systems.

Most web servers by default collect information about users using services residing on them. However, this information is often not sufficient for obtaining a clear picture about the behavior of the users. Thus, other approaches are to be used. The main idea is simply to log the required information while users' are using the service (*tracking*), and to analyze it and make relevant conclusions later.

In order to benefit from the information obtained from tracking, the data must be aggregated in one way or another, allowing for generalizations, although admitting some inaccuracies to keep it simple. One way to do that is to generate some user clusters, so that users who are in the same cluster are similar to each other (grouped by several criteria) but different from users in other clusters. The criteria for judgments have to be chosen carefully and the number of clusters should ensure that the groups are homogeneous. The study called *Cluster Analysis* provides theoretical basis on how to achieve all this.

Having aggregated several clusters of users, we can regard them as several "typical" users. Describing the typical users in each cluster and finding out more about their behavior gives very valuable information on where improvements in the software systems can be implemented, how the needs of those users could be served best.

The optimization of performance might require simulation of real-life scenarios, as explained earlier. By knowing how many users are in each cluster and how frequently they use the system, the simulation of the real-life environment can be performed much more easily.

Normally it is not possible to load a system by creating a number of simultaneous requests 'manually'. Thus, some automated approach is essential. The *load testing tools* serve this purpose. Roughly a half of the available load testing tools are commercial ones, the prices of some of them reaching tens of thousand of dollars. At the same time the

learning curve for most of the tools is quite steep and it is easy to lose a lot of time with the tools that are not capable to perform certain required operations, until one discovers their limitations.

The result of a load test allows one to judge how a software system performs under a certain load generated by many concurrent users using the system. This allows not only for the verification of whether the current implementation can fulfill the requirement of being able to serve a certain number of concurrent users with the desired level of service (measured, for instance, by the response time in seconds), but also has a number of other benefits for the different parties involved. The results of a load test allow management to know the capacity of their system and thus plan when the hardware is to be added; system administrators would find out whether the system crashes under the high load, or e.g. whether the number of the users accessing the system has to be limited so that the system would not crash, etc. Finally, developers could identify bottlenecks in the implementation and would become aware, the degree of optimization required. If there are bugs that reproduce only under high load conditions, running the load test allows them to be reproduced, so that developers can debug the system. Finally, the difference between a load test and a *performance test* (which is targeted to compare the performance of the system's modifications) is rather small and the load test can easily (depending on the tool used) be transformed into a performance test.

1.2 Objectives of this project

The goal of this project work is to analyze all the steps that need to be followed in order to perform a successful and effective load test. The major goal of a load test itself is to measure the performance of a web-based software system under a high load.

The study looks into several major approaches which could be used and many related techniques and aspects.

The stages of successful load test could be different depending on the approaches used and decisions made, but can generally be represented as follows:

- Understanding the goals which are to be achieved with the load test and the reasons to conduct it. (E.g. the goal might be to find out how many simultaneous users a system can serve within reasonable response time, which could determine the aggressiveness of a marketing campaign promoting the system)
- Collecting the data about users' behavior. (A load test simply simulates a high number of concurrent users using a software system, hence to construct a load test, the detailed knowledge about users' behaviors is essential)
- Analyzing how to perform the load test most effectively or efficiently depending on what is more appropriate and desired. (It must be understood, how important it is to be precise, fast or efficient with the results, whether or not spending a unit of resources justifies a slight increase in the quality of the results)
- Choosing corresponding load testing tools (There are plenty of the load testing tools available. What are the criteria to choose the right one? Is it reasonable to spend money on the commercial products?)
- Designing the test and configuring selected load testing tool accordingly (What exactly, in which sequence and how should be done to simulate the real-life conditions as good as possible with the information available and the load testing tool chosen)
- Performing the load test (What are the constraints which need to be satisfied to achieve correct results? Where in the network should the machine(s) performing the test be placed? How many machines are needed to perform a load test?)
- Validating and analyzing the results (Did it really work? Can the results be trusted? What could be concluded from the results? Did the system perform as has been expected? What are the bottlenecks of the system? What should be the next steps?)

This project work tries to describe general approaches for answering all these questions, as well as to analyze the most important aspects of each stage. A general approach is used where possible. As an illustration the software project "Warburg Electronic Library" (see next subchapter for more detailed description) has been taken, analyzed and load-tested.

This study describes and applies the techniques of cluster analysis in order to collect and aggregate data about users of a software system. This approach allows making conclusions which are beyond the scope of the load testing as such, but serve the same purpose – understanding the reasons why a software system performs the way it does under an increased load.

In case of the Warburg Electronic Library, the results of the cluster analysis give some ideas of how the system could be made more convenient for its users, what do the users look for when using the electronic library, how could the users be characterized by their behavior on the system, and, with the help of the load test's results, how could the performance of the system be improved.

The implementation of the actual changes of some software system and the assessment of their efficiency are beyond the scope of this project work.

1.3 Case Study: "Warburg Electronic Library" (WEL)

The Warburg Electronic Library is an interdisciplinary project carried out in conjunction with the art historians from Warburg-Haus Hamburg [http://www.warburg-haus.hamburg.de]. The first implementation of the project has been launched in 1997. Since 1999 the project has a web-based interface. The project's data is used in research and teaching. The system has currently more than 260 users, and classifies more than 10 000 pictures. Additionally, it holds other types of media documents, mainly videos. The design of the system and the major part of the implementation has been carried out by Dr. Hans-Werner Sehring. The project is built on the *Coremedia* platform. The following is the description of the WEL project taken from [http://welib.de/e-entry.htm].

> *The Warburg Electronic Library (WEL) is conceived as an open platform for interdisciplinary discourses: As a data-based working space linked to the Internet the digital library offers general access to multimedia documents and configurations adapted to the individual scientific requirements and research methods.*
> *In the course of a five-year research project conducted jointly by the Department for Information and Communication Technology at the Technical University of Hamburg-Harburg and the Research Centre for Political Iconography (University of Hamburg, Department of Art History) at the Warburg Haus, the "Picture Index of Political Iconography" was transformed into a digital multimedia library.*

The practical part of the project has been based on the analysis of the WEL system. The real usage data has been taken from WEL logs. This data has been used for the analysis of users' behavior, clustering users according to their behavioral patterns, suggesting improvements, performing the load tests of the WEL platform, and, finally, analyzing the results of the system performance.

The system suits the objectives of this project, because it is a web-based application with sufficient complexity, sufficient number of users who can access the system simultaneously. Moreover, the system has been tracking the data about the users' behavior over more than 4 years. The information contained in the log files is sufficient for all objectives of this project work.

The following are look-and-feel screen shots of the Warburg Electronic Library briefly giving an idea of platform's functionality.

The following screen shot displays a list of libraries one specific user has an access to. By accessing links on the left hand side menu the user can look for some concrete content using the *Search* functionality, edit own account's settings (e.g. change own password), create a new library, browse further to one of the libraries, log off, etc.

Figure 1: Screen shot of the WEL libraries listing page

The screen shot below displays a sample *Classifier* page. Classifier page displays objects (of different types) having some logical connection to it (please refer to the Figure 3 for a database model of WEL). A user can browse further to one of the related objects of a classifier, edit this object (if s/he has corresponding roles and rights), etc.

Figure 2: Screen shot of the WEL classifier page

The WEL system is organized in such a way, that every page (with very few exceptions, like, for instance, log-in page or user's settings page) is determined by the object id. Objects can be of several types:

- Classifier Index - represents main listing of the "collections" of the electronic library entries. An affiliation table links *Classifiers* into "directories" / "collections".
- Classifier – a categorization of objects, "category" (e.g. Money).
- Hierarchical Extent – represents additional hierarchical structures (e.g. "Music" is a hierarchical extent, might organize all music related entries).
- Card – an actual content of the electronic library, e.g. pictureCard, movieCard, textCard.

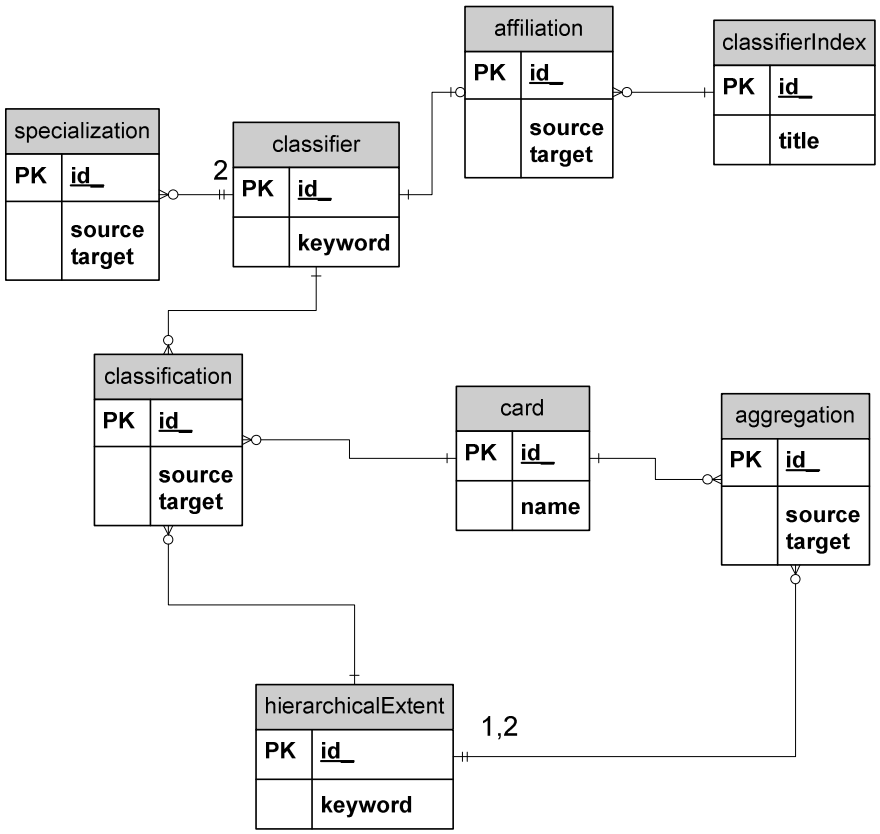

Figure 3: Database Model of the Warburg Electronic Library (simplified)

2. Approaches for the User Cluster Analysis

This chapter describes how to get the data about web application users, a theoretical basis of the *User Cluster Analysis* and how has the user cluster analysis been applied to WEL.

2.1 Collecting Data about Usage of the System

2.1.1 Tracing

There are several ways to collect data about the usage of web applications.
If a web application does not log specific information about its users, most likely, there is still a way to have a look at the default usage log files(unless disabled) and retrieve some information about it.

The most popular WEB Server, Apache, (with the 68% of the market share (according to [13])) has a possibility to log certain information about the incoming http-requests.
The default logging format (according to [1]) is :

%h %l %u %t \"%r\" %>s %b

where

%h	-	the IP address of the client (remote host) e.g. *127.0.0.1*
%l	-	the "hyphen" in the output indicates that the requested piece of information is not available.
%u	-	the user id of the person requesting the document as determined by HTTP authentication. e.g. *admin*
%t	-	the time that the server finished processing the request.
%r	-	the request line from the client. e.g. *GET /rg/login.jsp HTTP/1.0*
%>s	-	the status code that the server sends back to the client. e.g. *200*
%b	-	indicates the size of the object returned to the client, not including the response headers.

Hence, Apache logs some essential things by default, like: request time, IP address, HTTP authentication user id (although, the HTTP authentication is relatively rarely used), the path of the file accessed. However, that might be not enough in order to collect information about user's behavior. First, the HTTP POST/GET parameters might contain valuable information about the resources being requested. Secondly, the IP address cannot identify a user of the system uniquely, because some of the users might use proxy-server, so that only proxy-server's IP would be logged.

So, the information which could be found in the default Apache's "access.log" file does not have a way to identify a user precisely (a combination of the IP address and

a “User-agent” would be a slightly better approach) from the application logic point of view.

Thus, tracing is insufficient for the purposes of this project work, unless some constraints are met, for example, if the IP address identifies a logical application user uniquely, etc.

2.1.2 Tracking

Another way to collect information about users is to actively log the information which is needed for the later system usage analysis. The logging might be done in some core library, which code is being executed with each http request, and also in some decisive parts of the code.

The following are the minimum information needed to be logged in order to be able to trace users’ behavior in the system:

- User identifier
- Exact time of the request
- Identifier of the requested resource
- Important parameters bypassed (POST and GET)

As for the WEL log files, all above mentioned information could be found there. The WEL application is organized in such a way, that a user simply accesses different objects (of different types) by specifying the object ID as a GET parameter of the http-request. The requested path mostly stays the same.

2.2 Cluster Analysis

2.2.1 Introduction into Cluster Analysis

Cluster analysis deals with grouping different objects (e.g. users) in such a way that the objects from the same group are to the greatest possible extent similar to each other and different from the objects from different groups. Those groups or categories of objects are called *clusters*. The criteria for similarities or differences are called *variables*, *observations*, or *dimensions*.
The conduction of cluster analysis consists of several major steps:

- Selection of the relevant variables
- Selection of the distance measure
- Selection of the clustering algorithm
- Determining the number of clusters
- Validation and the analysis of the results

In the simplest case there is just 1 variable for which classification or cluster analysis is done. But normally there are more. Variables are selected logically, having the goal of achievement an adequate characterization of the objects in mind. The approach used for the selection of relevant variables should strive for the minimization of their number. But the variables used might be measured in different units. For example, it is desired to cluster people by 3 variables, say, age of a person and the observation whether a person smokes (the value for it is, say, 1 or 0) and person's high school's average grade (e.g. measured from 1 to 5). Obviously, these variables are measured by different units, so before any algorithm for calculating the *distance* between 2 objects (measuring how different or similar those 2 objects are) can be applied, the value of each variable for each object (in the given case, for each user) has to be "normalized" according to the variations of the variable. This is done by dividing the value of each object's variable by the standard deviation of the variable:

$$\delta = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2}$$

where

δ	-	standard deviation
N	-	number of objects
x_i	-	value of the variable for the object nr. "i"
$\bar{x}$	-	arithmetic mean value of the variable

2.2.2 Distance Measure between Objects

When variables' values are normalized for each object, one of the algorithms for the calculation of the *distance measure* between 2 objects can be applied. The following are some of such algorithms:

- Euclidean distance - probably the most widely used one. It is simply the geometric distance in the multidimensional space. Computed as:

  $$\text{distance}(x, y) = \left(\sum_i (x_i - y_i)^2 \right)^{1/2} ,$$

  where "x" and "y" are 2 objects, between which the distance is being calculated.

- Squared Euclidean distance – placing progressively more weight on the objects which are further from each other. Computed as:

  $$\text{distance}(x, y) = \left(\sum_i (x_i - y_i)^2 \right.$$

- Chebychev distance – the distance is the greatest difference between dimensions. Might be used if the difference in one of the dimensions is considered not to be compensated by the similarities in other dimensions. Computed in the following way:

  $$\text{distance}(x, y) = \max | x_i - y_i |$$

- Percent disagreement – could be used in case dimensions are of the categorical and non-analogue nature. Computed as:

  $$\text{distance}(x, y) = \text{Count} (x_i \neq y_i) / N ,$$

  where "N" is the number of variables.

Each of these distance measure algorithms is particular useful under special conditions, depending on the nature of objects and variables used for categorization.

2.2.3 Clustering Algorithms

The following figure characterizes clustering algorithms.

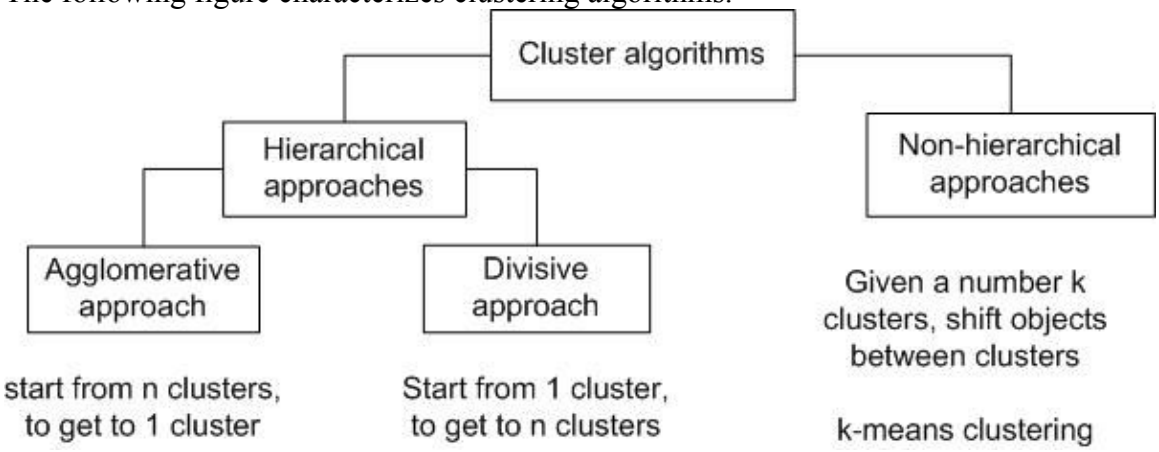

Figure 4: Characterization of clustering algorithms

The simplest algorithm to implement in case of the hierarchical approach (when the number of clusters in unknown (that would normally be the case when trying to classify users of the web application)) is the *agglomerative clustering*. That algorithm can be briefly described by the following steps:

- place each object into a separate cluster
- start unifying clusters according to a certain "combining method"
  - assess the differences between objects within each cluster
  - determine the number of clusters according to the principle: if there is a sudden increase in the error measure, stop minimizing the clusters' number; otherwise continue unifying clusters

There are several ways how to combine clusters using agglomerative clustering algorithm, among them:

- Linkage method
  - single linkage (minimum distance; tendency for large clusters)
  - complete linkage (maximum distance; tendency for small clusters)
  - average distance (those clusters are combined, whose increase of the average distance between objects in the combined cluster would be lowest)

- Ward's method – most popular method (tendency for equal size clusters). The algorithm includes the following steps:
  - computing the sum of squared distances within clusters
  - combine clusters where the minimum increase in the overall sum of squares would be minimal

- Centroid method – the distance between 2 clusters is defined as the difference between *centroids* (cluster's average point in the multidimensional space)

As in case of selecting the distance measure algorithm, there is no single correct answer for the selection of an agglomerative clustering algorithm.

3. User Clusters in the WEL System

This chapter explains how the cluster analysis has been conducted for the Warburg Electronic Library (WEL) system and which results have been achieved.

3.1 Preparation for the Cluster Analysis

3.1.1 Collecting the data

First of all, the performance of the WEL cluster analysis is for the needs of the load testing. The load test has to simulate the http requests coming from many online users. The simulated users should act in the way the real users are normally acting. For example, some group of users generates on average, say, 15 page views, uses search once, downloads 2 full size pictures and spends the rest of the browsing path in order to find those pictures. Let's assume that kind of users constitute 20% of all incoming requests on the WEL system. This means that during the load testing, $1/5^{th}$ of all requests should follow the same "browsing behavior". Now the task is to acquire the corresponding data about users' behavior. For this purpose a program parsing WEL log files and interpreting their data has been written. The figure below sketches its functionality.

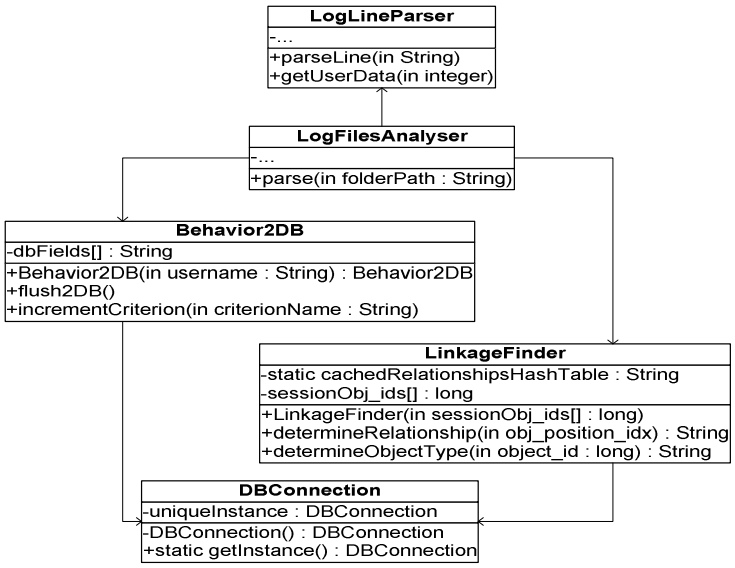

Figure 5: Class diagram of the program parsing WEL log files

3.1.2 Selecting relevant variables

Having the goal of generation user clusters for the conduction of the load test later in mind, the variables for the cluster analysis have to be chosen. All of the variables chosen represent different average values per session, because the goal is to find users acting in a similar way. The following 10 variables have been chosen:

- average number of objects accessed per session
- average number of following the browsing links per session
- average usage of search per session
- average usage of search for a card object per session
- average usage of the links on top of the card's page
- average rate of re-accessing an object per session
- average rate of re-accessing a card object per session
- average rate of re-accessing a classifier object per session
- average rate of accessing a classifier-index object per session
- average rate of accessing a card object per session

Those variables are going to reveal the behaviors of users during their typical session. Presumably, there are some behavioral patters, for example, more experienced users would use the search of a card object, whereas less experienced one, would use the default "search for a classifier" functionality, without changing the type of an object they are looking for, to something else.
Still the selection of the variables is somewhat subjective and has been also based on my experience of trying different options.

3.2 Performing the Cluster Analysis

To determine the distance between 2 users' data the Euclidean distance formula has been used. This is a logical choice considering that relatively high number of variables has been used, that the variables are of a very different structure and that the objects (users) are also very different. Further, the hierarchical (determining the number of clusters trying to find the best solution) agglomerative (minimizing the number of clusters starting from N clusters) approach has been used. There has been no expected result, so the number of clusters had to be determined by the cluster analysis itself. Since there has been no suitable (free and simple) software found, a Java program has been developed to conduct the cluster analysis. The agglomerative approach is the easiest one to implement; therefore it has been chosen for the cluster analysis. In order to combine two "closest" clusters one of the linkage methods, namely, the average distance principle (combining clusters with the lowest increase in the clusters' average distance between objects) have been applied. This choice has been made taking into account high degree of the differences between the objects.

The class diagram below depicts the structure of the developed software, which was used to conduct a cluster analysis for the WEL users.

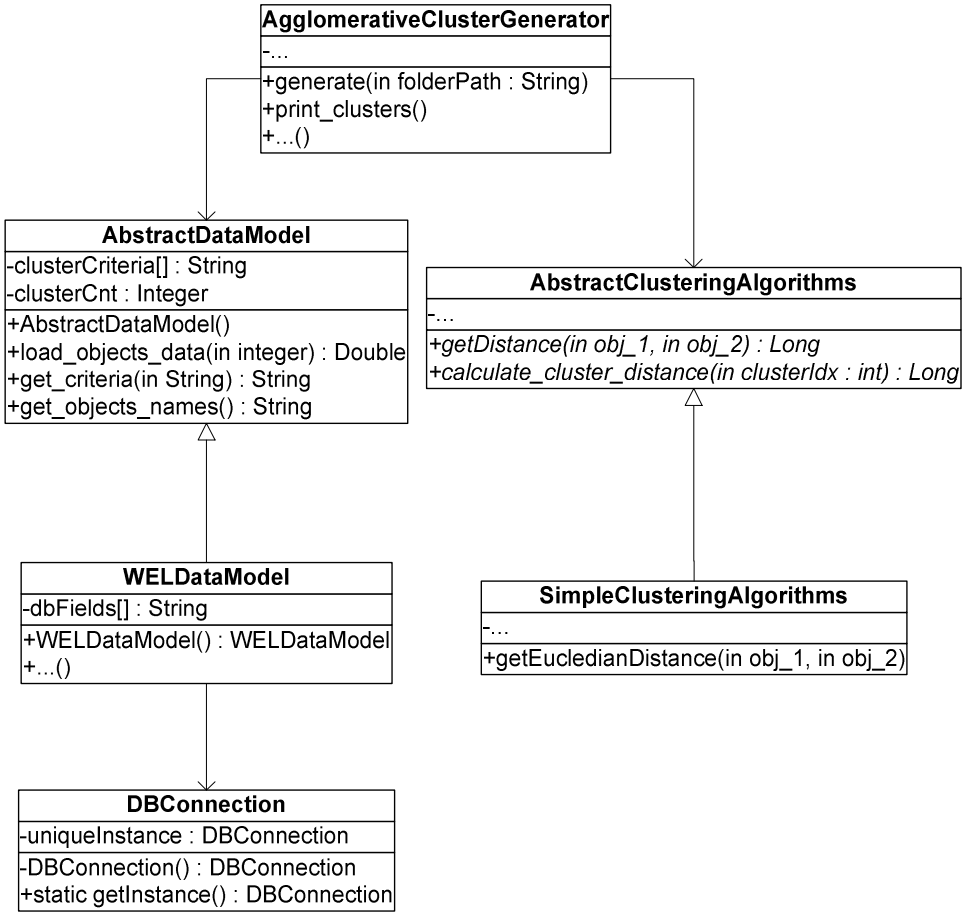

Figure 6: Class diagram representing the structure of the program "AgglomerativeClusterGenerator"

The following figure represents the results derived from the execution of the program, namely, the dependency between the "error measure" (sum of squared distances within clusters) and the number of clusters, in which the WEL users have been placed according to the algorithms and approaches mentioned above.

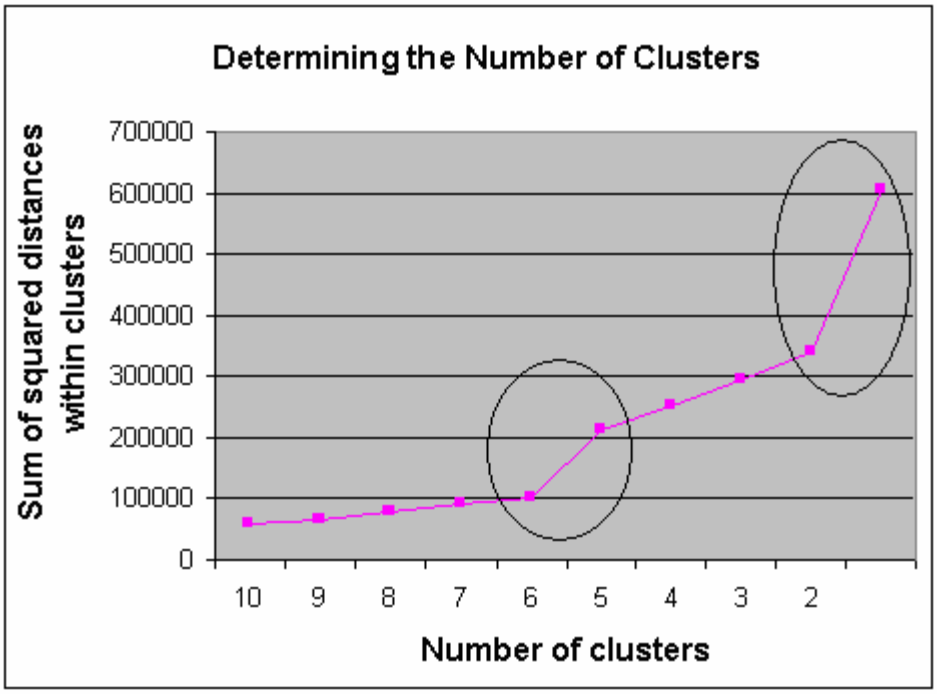

Figure 7: Determining the number of clusters

As can be seen from the figure 7, there are two "sudden increases" ("elbows") of the error measure. The increase of the error in case of moving from 1 to 2 clusters is more "sudden". It should be stressed, that this figure represents absolute numbers and not the relative ones, so the first impression might be a bit misleading.

The following table shows the actual values revealed.

#Clusters	Sum of squared distances within clusters	Increase in error when decreasing # clusters
10	58189	13%
9	65846	18%
8	77760	16%
7	90280	11%
6	100362	113%
5	213757	18%
4	253067	16%
3	293873	15%
2	338631	79%
1	606185	

Table 1: Clusters' error measures

The following figure explicitly shows the increase of the error when decreasing the number of clusters by 1. The maximal increase of an "error measure' happens when the number of clusters is being decreased from 6 to 5. Thus, according to the principles described in the previous chapter, 6 clusters should be taken as an optimal value.

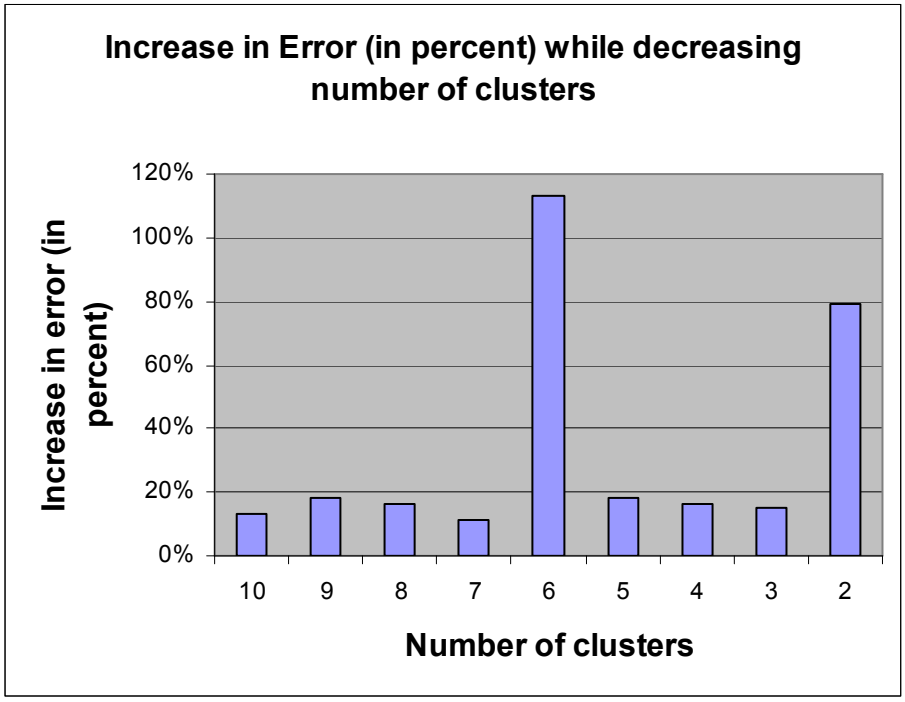

Figure 8: Increase of error when decreasing the number of clusters

There are more reasons why to choose 6 clusters, instead of 2. First, of course, is that the "inaccuracy" of combining different users into 6 clusters is about 3 times less than combining them to the 2 clusters. Secondly, having 6 clusters also allows eliminating possible negligible groups of users which disturb the average values by being too different.

3.3 Analyzing the Results of the Cluster Analysis

The following table contains the most important characteristics of the 6 clusters retrieved in the cluster analysis. (See Appendix 2 for more detailed information.)

Cluster ID	**1**	**2**	**3**	**4**	**5**	**6**
Description	**Users looking for something concrete**	**Browsing users**	**Negligible**	**Neglig.**	**Neglig.**	**Neglig.**
Percentage of users	54,8	39,2	2,3	1,5	1,5	0,8
Number of users	144	103	6	4	4	2
Percentage of all objects accessed	**37,5**	**53,3**	**1,3**	**5,7**	**0,9**	**1,3**
Percentage of all objects re-accessed	25,2	67,7	0,7	4,7	0,6	1,1
Total objects accessed	6561	9329	219	1002	159	233
Average number of sessions	7	7	5	3	1	2
Objects accessed per session on avg.	**7**	**12**	**7**	**78**	**40**	**58**
Search used per session	1	0	2	1	0	1
Objects re-accessed per session in avg.	1	3	2	21	11	18
-Classifier index	1,46	3,17	1,68	5,14	10,25	8,83
-Classifier	4,6	7,05	3,16	34,49	26,5	45
-Card	0,98	1,18	1,91	29,93	1,25	4,67

Table 2: Main characteristics of the clusters

For the ease of further analysis and load testing, 2 largest clusters (the 1st and the 2nd one) only, which constitute about 94% of all objects requests, are going to be taken into consideration. The remaining 4 clusters are going to be neglected, because they represent very rare users which are not important for the purposes of the load testing.

Users from both 1st and 2nd clusters have rather short browsing paths. This is because of a high ratio of users who are new to the system and make very few requests to check out what does the system offer, so the "browsing path" of the "active users" might be longer.

The cluster 1 can be conventionally described as "Users looking for something concrete", because the users' browsing path is quite short – on average they access just 7 objects, use search once and access the "Card" object once. After they found what they wanted, they download one or two "Card" objects and leave. The 2nd cluster can be conventionally called "Browsing users". These users access about 12 objects on average, use almost no search (which is a more advanced functionality), but spend some time in the WEL system browsing, going back to the libraries index choosing another category or re-accessing the category their have already accessed.

The following sequence diagram represents typical (slightly shortened and simplified) browsing path of a typical user from cluster 1. This sequence diagram helps to understand, which database transactions lay behind the user's requests. Each of the interacting objects needs to execute several database queries in order to display the resulting page.

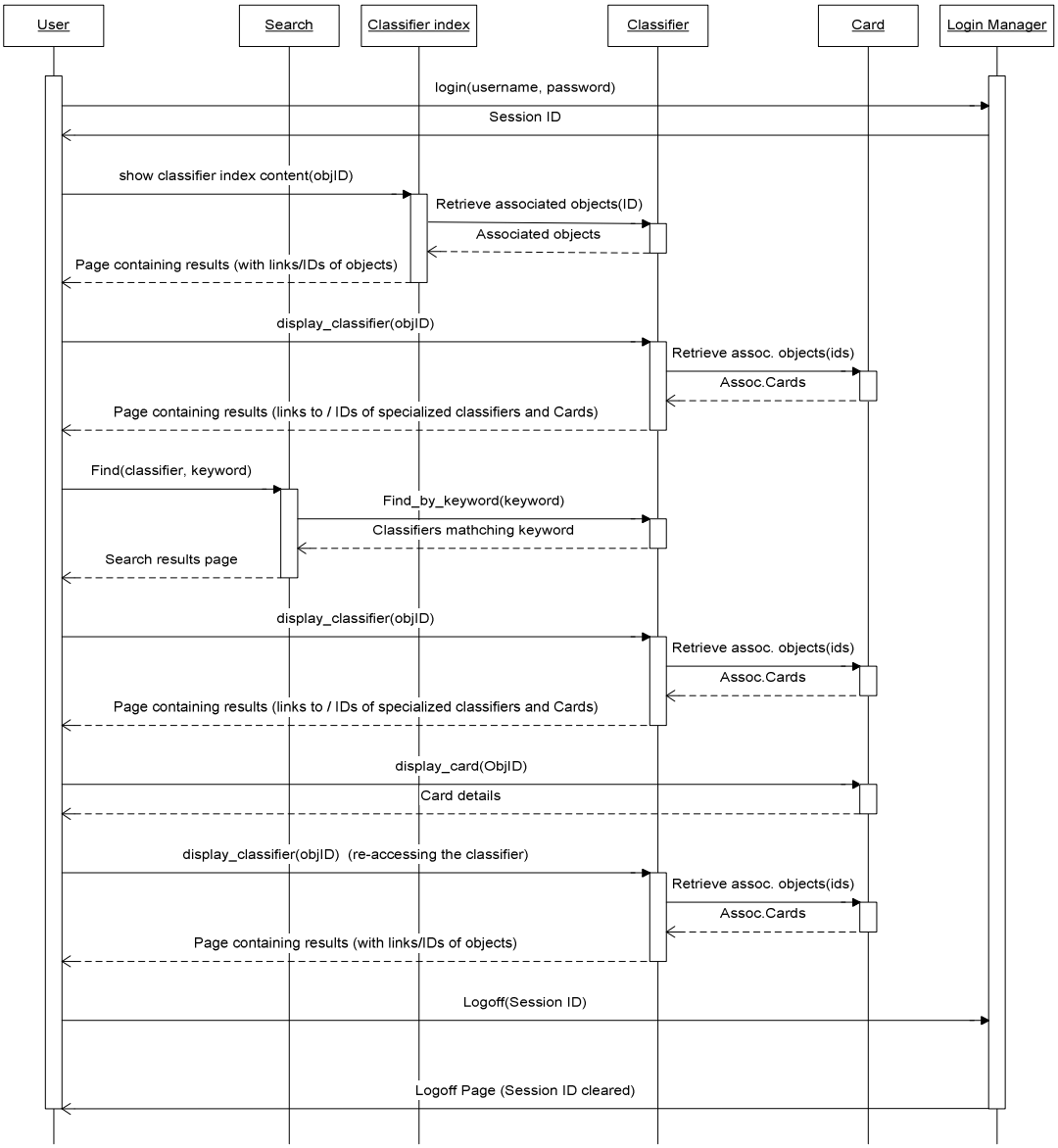

Figure 9: Sequence diagram depicting major actions of users from cluster 1 during a single session.

4. Load Testing Tools

This chapter gives an overview of the *load testing tools*, their configuration and usage, and explains the choice of the load testing tool for the WEL application.

4.1 Introduction to the Load Testing

The *load testing* tests system's performance under a high load generated by simulating a number of concurrent users accessing it. Load testing does not have an intention to verify, whether a web application is functional (that kind of test is called *functional testing*, and implies checks like whether all links work, whether all input forms are protected from a crap being entered etc). However, during the preparation to the load testing some of such problems might be revealed. The load testing differs from the *performance testing* in the way, that it does not compare the behavior of the application under different settings (e.g. comparison of an earlier version towards a new one). During the conduction of the load test, the greatest number of users, which the platform is capable to serve within a reasonable response time, can be revealed.

The simulation of a high number of simultaneous users should be executed as close to the real conditions as possible. Requests coming from real users are different, because of many reasons, which include the following:

- Users behave differently (e.g. different frequency of accessing different resources, different duration of time spent using the application per session)
- Users have internet connections of different speeds
- Users use different browsers, different versions of the same browser with different settings (e.g. image caching), etc
- Users access a web application from different geographical places with IP addresses of different ranges (which might mean the application generates the output in several languages / with differentiated content)
- Users might have different settings in a web application (e.g. number of email accounts configured or number of news blocks to be displayed)

All those variations and the high quantity of their combinations make it very difficult to prepare the load test which is going to be close to the real-life conditions, and even more difficult to have a tool which would be able to prepare and conduct good load tests close to the real-life conditions for every possible web application. One argument to support this is the fact that it would be difficult for an automated system to predict users' behavior in a new sufficiently complicated application unless the experience with similar frameworks already exists (e.g. user might misunderstand the meaning of some link and will try to find another way to reach the resource s/he wants, which might have negative or positive impact on the performance of the system).

Thus, "manual" work is most likely needed in order to conduct a good load test. But usage of one or several available tools is definitely not only a great help in that, but also a necessity, because it is impossible to simulate sufficient number of simultaneous users "manually" in most cases.

4.2 Categorization of the Load Testing Tools

There are many load testing tools available both commercial and free ones. They differ as much as programs for a non-trivial and non-standardized area from different producers might differ. Some of the differences have been categorized into the groups listed below.

- Free or commercial
  There is a high number of both (at least 50 in total). Normally, commercial solutions have more convenient GUI and have nicer reporting, but the functionality of the free ones is normally sufficient as well. Moreover, non-commercial load testing tools are almost always open source, so they might be changed or enhanced if needed.
- Interface
  Different tools provide different user interfaces / presentation layers. Among them: Graphical User Interface (most of them), command line interface (e.g. httperf).
- Configuration
  Four different types of configuring the test (combinations are possible and very likely) might be distinguished:
  - Configuring the test using graphical user interface (GUI). Normally, that kind of tools would simply provide a way to specify URLs to be accessed, as well as the way to configure additional things (e.g. random parameters, requests depending on responses and so on).
  - Tracking the user's actions in a browser. User can simply browse to the server to be tested and the testing tool will track the browsing path and all the parameters and actions a user would be doing. Most likely, the tracked data will be used as a template which could be changed later (e.g. using some programming language).
  - Tracing user's actions by looking at the network traffic. In this case, user may use several browsers or there could be several users as well.
  - Writing a program in some programming language – existing or artificial. That gives a great flexibility, but makes it more difficult to quickly construct or change test plans.
- Availability of distributed testing
  A single machine might be not enough to handle enough requests to a server. It is essential that the machine performing the test had enough resources to cope with the processing of the data; otherwise the results might be wrong. So, some tools provide a possibility to perform a test using several machines (normally, one being a "master" and several being "slaves").
- Execution
  Most commonly the intermediate results of going on test are displayed in the interface the tool (e.g. as a command line output or in the GUI). Another aspect of the execution is *how* exactly the outgoing http requests are made. There are 2 options: either by sending the http requests directly to the server (by connecting to

the server, possibly via a proxy server) or via a middleware standard browser, which can interact with the tool (e.g. Watir [22]). In this case the browser is to execute http requests in the most common way. This has several benefits, because the resulting page is being displayed exactly in the way an end-user would normally see it (with a real browser). The JavaScript will be executed as it should and the test can even be configured to interact with JavaScript (e.g. pop-up windows, JavaScript alerts) on behalf of a user. This would not be always possible with other approaches. Also, that kind of approach allows a user behind a proxy server to try out the tool faster, because the browser typically already has proxy settings configured. But it has also negative aspects and the major one is performance. That kind of tools might be used if they allow distributed testing and there are a lot of computers available or in case the testable server's performance is really weak, so that the test does not require too many concurrent users.

- Performance
  The load testing tools can differ with regard to performance quite a lot. Adding one more simultaneous thread / user can require considerable amount of resources. (e.g. testing tools displaying the test in a browser definitely require much more physical resources on a testing machine(s), comparing to the command line ones)
- Reports and metrics
  Usually, commercial tools are much better in this aspect, making it convenient to use the information obtained during the test, which might be not necessarily true for the free tools. Among the options how to report or measure performance, there are the following: number of various figures, which could be constructed by the tool, whether the response times can be normalized by their standard deviation by the tool, whether it is possible to adjust which information is desirable etc.
  The measurable parameters may include:
    - average, max, min, deviation of the response time per page type
    - number of failures (e.g. 500 - server not responding)
    - throughput - number of requests handled by server per minute
- Ease of use and steepness of the learning curve
  Most of tools require quite a lot of learning before any reasonable test could be performed. One reason for this is namely the great variety of them and different approaches used. Another reason is the absence of standardization and even terms used. On another hand, the test of non-static content of some relatively complicated web application is simply not a trivial task, because there are too many aspects which might differ from one application to another.
  Of course, it is normally easier to start with a tool having test's configuration in a known programming language, than learning a made up language.

4.3 Configuration

Different load testing tools provide different ways how the configuration of a load test could be done – for instance, by writing a program (e.g. in Java) or via some GUI. But whichever way is used, there is one underlying principle: a testing tool should be able to simulate real-life conditions to the needed extent.

Let's have a look at several load testing tools in order to understand and compare different configuration's issues.

4.3.1 WebUnit

WebUnit is a free open source Java program, initially targeted to functional testing, but can also be used for load and performance tests. The responses from the server are not simply logged to a file (or temporarily to the memory), which is most commonly done, but are actually displayed in a browser window(currently, only MS Internet Explorer is supported). This has a number of advantages, because it makes the test closer to the real-life conditions. For example, the delay required for a browser to display the retrieved page could also be considered directly, which is not the case with the other kinds of tools. The following quote from the WebUnit's documentation explains how it is done:

> *The ExplorerWebBrowser implementation is done using Microsoft's Java-COM VM. The actual Explorer COM API is used. The WebUnitCore will actually exec the MSJava VM and communicate with that process(s) over its I/O streams. A simple Java I/O – RPC mechanism is part of web-unit allowing this communication to be efficient and not require any socket communication.*

The test is being entirely controlled by a Java program, which might be seem to be time consuming to configure, but at the same time it provides a great flexibility.

The sample script listed in the Appendix 1 is an illustration how a load test could be performed with WebUnit. It goes to the Warburg Electronic Library site, logs in, and browses around a bit, measuring the response time. The program consists of 2 classes:

- SimpleLoadTest – main program, which launches X threads 'WELThread'
- WELThread – a thread, which simulates a user on the WEL. First, it goes to the WEL start page, then clicks to the image-link, logs in, accesses list of libraries, picks up one of them (by clicking on its name) and, finally, accesses on of the subcategories (a "classifier"). In order to browse, the configuration uses actual textual strings (e.g. links' "names") displayed to a user in a browser window, names of the HTML variables (and sometimes names of the image files (which could be escaped by changing server side code)). This is different from what most of the load testing tools do, namely, require fully hard coded URLs.

The following few lines of code illustrate main principles of the scripts based on WebUnit. The piece of code accesses few pages of the WEL system. As can be noticed, there are no hardcoded URLs in the code (except for the very first one, which specifies the start page), but they can be retrieved from the pages' sources instead (e.g. by specifying the image-link's parameters, such as HTML tag's id, image's filename etc).

```
// the following line will initialize the IE browser
wc = DefaultWebFactory.getFactory().newWebClient();

// request the WEL's start page
register_pageview("Start");
wc.openPage(sUrl);
// retrieve the page data
WebResponse currentPage = wc.getCurrentState();
// measure & log the response time
register_pageview("Show Start Page");

// find the log-in link (the image 'zugang.gif') & click it!
currentPage.getLink("zugang.gif", WebTag.FIND_CONTAINS).click();
// retrieve the page data
currentPage = wc.getCurrentState();
// measure & log the response time
register_pageview("Go to Login Page");
```

4.3.2 JMeter's configuration

JMeter is a free open source Java testing tool, which among other things allows performing load tests. It supports both non-GUI and GUI modes. The configuration of a test plan can be done by adding various "elements" to a test, such as *Listeners* (used for reporting and the analysis purposes), *Timers* (used for scheduling or delaying events), *Thread Groups* (concurrent executions), *Assertions* (to check whether the received response is what has been expected) and others – to control the flow of a test and specify the settings of the requests.

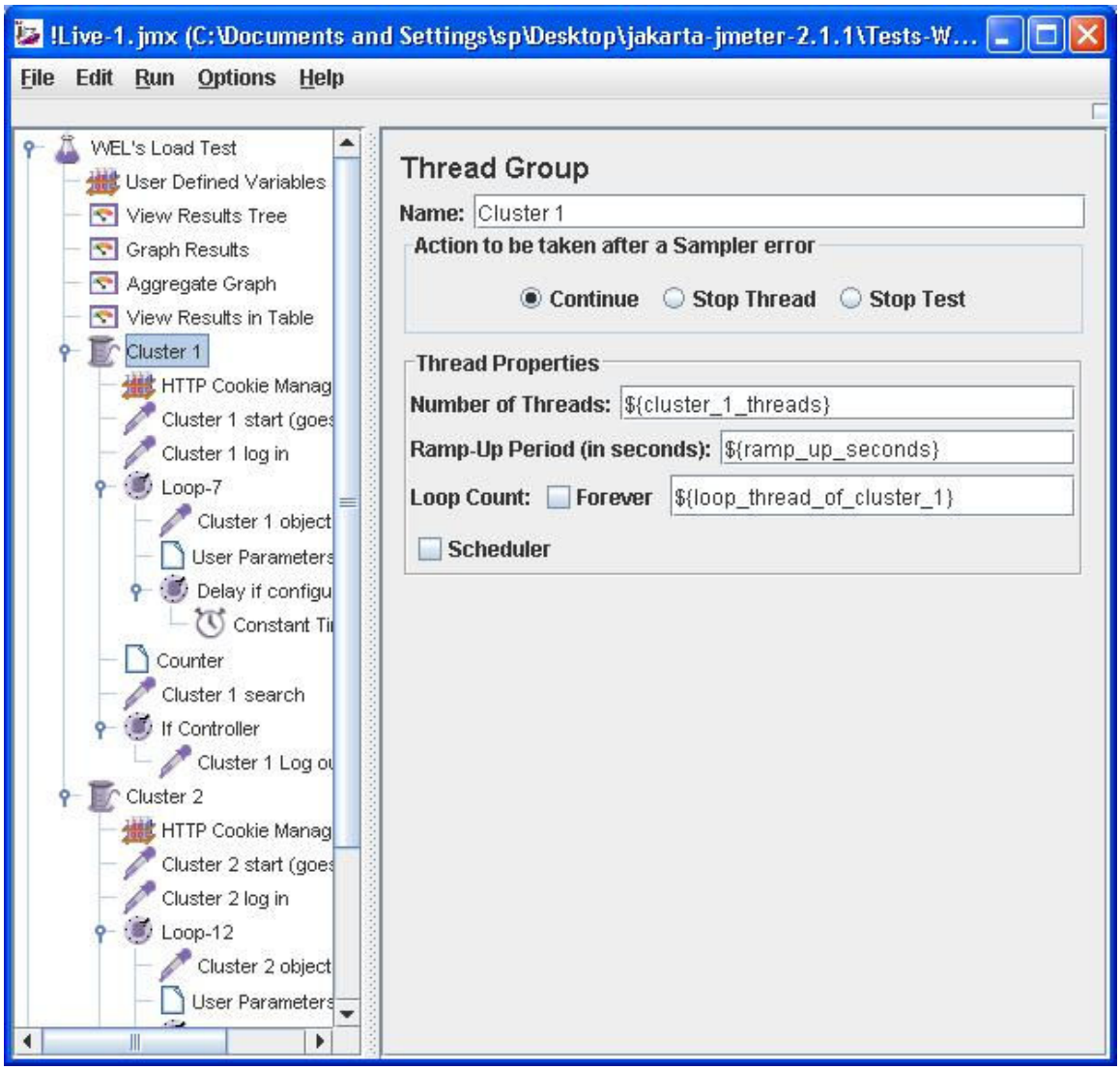

Figure 10: The screen shot of JMeter's configuration of cluster 1 threads

Let's have a look at how different test plan configuration issues can be addressed with JMeter. Often each thread in a test plan must make very many similar requests to the server, which might differ just by a part of arguments or a part of the requesting path. In this case the addition of e.g. one thousand different http requests for each thread especially in the GUI is not feasible. To address this kind of issues JMeter provides a possibility to load data from external data files. This can be done by one of the following ways:

- using the function *_StringFromFile*, which when called returns the next line from a file. Then the returned string can be parsed by other means.
- using the function *__CSVRead*, which can return the value contained in a CSV (coma separated values) file in a specific column on a specific line.
- using the special configuration element "*CSV Data Set Config*", which allows for extracting values from a CSV-like (coma separated values) file.

4.4 Load Testing Tools for "Warburg Electronic Library"

After checking multiple tools JMeter has been chosen to perform the load testing of the Warburg Electronic Library. This is because it is a good combination of relative simplicity of configuration and flexibility in the functionality.

According to the performed cluster analysis (please refer to the earlier sections) the load test should simulate the behavior of two types of users. The information about the differences of those two groups is summarized in the following table (NB! All the values are rounded).

	Cluster 1	Cluster 2
Description	**Users looking for something concrete**	**Browsing users**
Percentage of all requests made	41%	59%
Search functionality used per session	50%	0%
Number of objects accessed per session	7	12
- Access of Classifier per session	1	3
- Access of Classifier-Index per session	5	7
- Access of Card per session	1	1
- Access of Hierarchical Extent per session	0	0
Number of objects re-accessed per session	1	2
- Re-accessing Classifier per session	1	1
- Re-accessing Classifier Index per session	0	1

Table 3: Aggregated data per cluster used for the load test

The following diagrams depict the browsing path of the users from each of the clusters, used in the load test configurations.

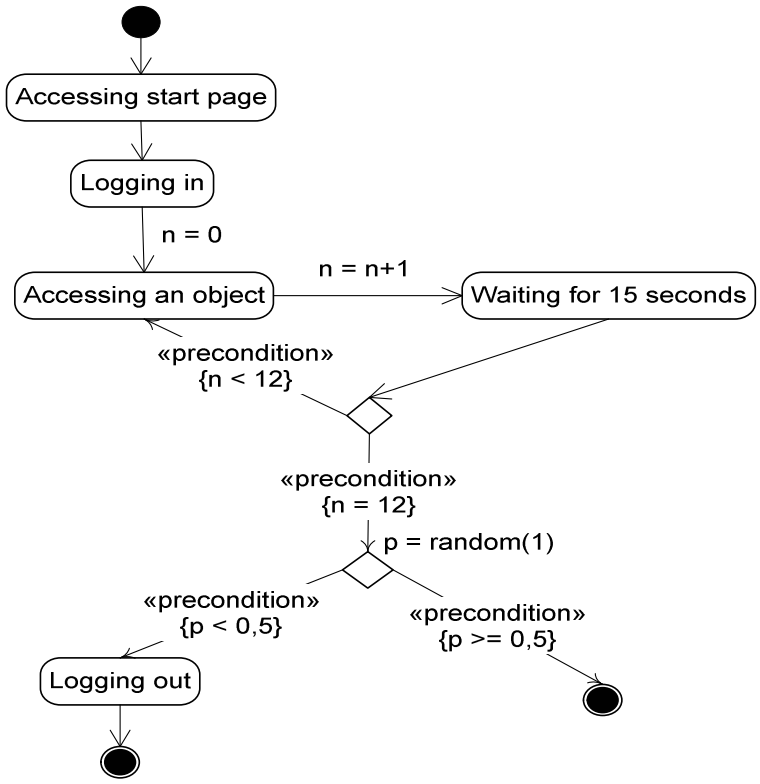

Figure 11: Diagram displaying browsing path for cluster 1

From the browsing path point of view, the users from cluster 2 are different from 1st cluster ones mainly because they do not use search, and access more objects per session.

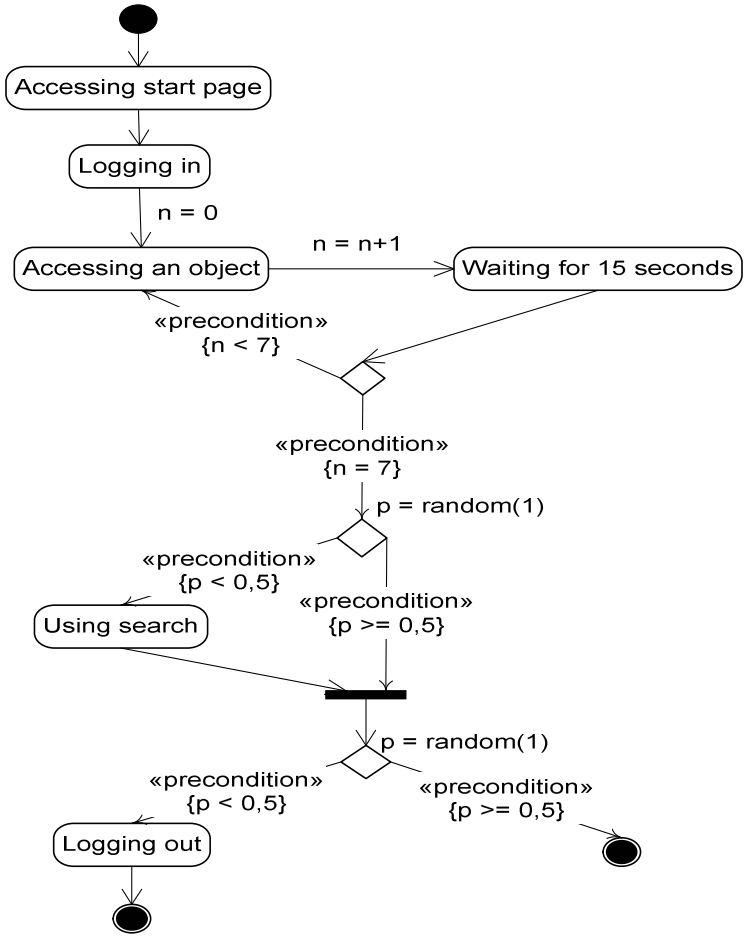

Figure 12: Diagram displaying browsing path for cluster 2

The load test has been configured according to the diagrams above. The delay between requests has been measured by using the data from WEL log files. Random objects ids of certain types have been put to several CSV files. This way the behavior of users is being simulated as close as possible to the real-life conditions. The following chapter contains results of the test performed.

5. Performing the Load Test

This chapter describes various aspects of the conduction of the load test as well as the analysis of its results.

5.1 Designing a Load Test

Before starting to design a load test, one should be clear about the goal which is to be achieved. The questions like why the load test is needed, what exactly is to be tested, how important is it to be as close as possible to the real live conditions, what depends on the results of the test, and so on. The next steps depend on how these questions are answered.

Generally, the purpose of a load test is to measure the performance of a system under a high load. The underlying goal of conduction of a load test is to improve performance, in case the load test shows that it is needed.

There are 2 major use cases: testing an existing system and testing a new one. In case of testing a new system, the goal is to find out, whether it can hold the needed number of concurrent users. The behavior of the users should be rather guessed, based on the subjective criteria. In the second case, when testing an existing system, the data about the behavior of existing users should be analyzed. Already the aggregation of the usage statistics will reveal many niches for optimization. Some functionalities might appear to be more popular, whereas other less popular, than has been expected. Further step might be to cluster users into groups. This has several benefits. Among them there are the following: better overview about the users of a system, a possibility to design modifications by predicting changes in the quantities of different types of users, limiting the inaccuracies of using the average numbers.

After the information about the behavior of users has been collected, it is time to choose a proper load testing tool. The selection depends on the complexity of the users' behavior and many other things. If the requests which have to be simulated are very few (e.g. 5 different URLs which each user accesses), then the most simple tools should be used, if the reporting they provide is sufficient. In more complicated cases, the performer of a load test should use more flexible and "mature" tools. In the very rare cases it might be suggested to use open source tools so that a load test performer had a full control over the test and the testing tool, modifying the source code if needed. But normally, the functionalities provided by a load tool are enough.

5.2 Execution of a Load Test

The location of the machine(s) performing a load test in the network must be chosen according to the goals of a load test. If the intention is to measure the response time from the end-user's perspective, the machines are to be placed further away from the server(s) which is/are being tested. Nevertheless, the network's throughput should not become a bottleneck, because it is solely the performance of the system what is being tested. There are other (normally simpler) ways to find out whether network's throughput is sufficient. Thus, the machine(s) performing a test must be placed as close to the system as possible to avoid limitations set by the network.

During the conduction of the load test some time is needed in the beginning for so called "ramp up phase". The ramp up phase is a period of time during which the testing tool starts launching all of its threads. Threads normally should not start sending requests all at once, because the intention is to simulate real users accessing the system independently at the random points of time.

The typical response times during the conduction of the load test could be seen from the figure below. The ramp up phase with shorter response times could be easily distinguished. The ramp up phase is not particularly important, unless the increase in the response times is too rapid, which means that the system should probably be tested with the smaller number of simultaneous users. As can be seen from the figure, there are always some cases, when the response time is low, even when the overall load on the system seems to be high. This is because of the presence of the completely static pages, which are processed very fast, because they are not related to the bottle necks of the system. (This can differ depending on the internal structure of the system tested.)

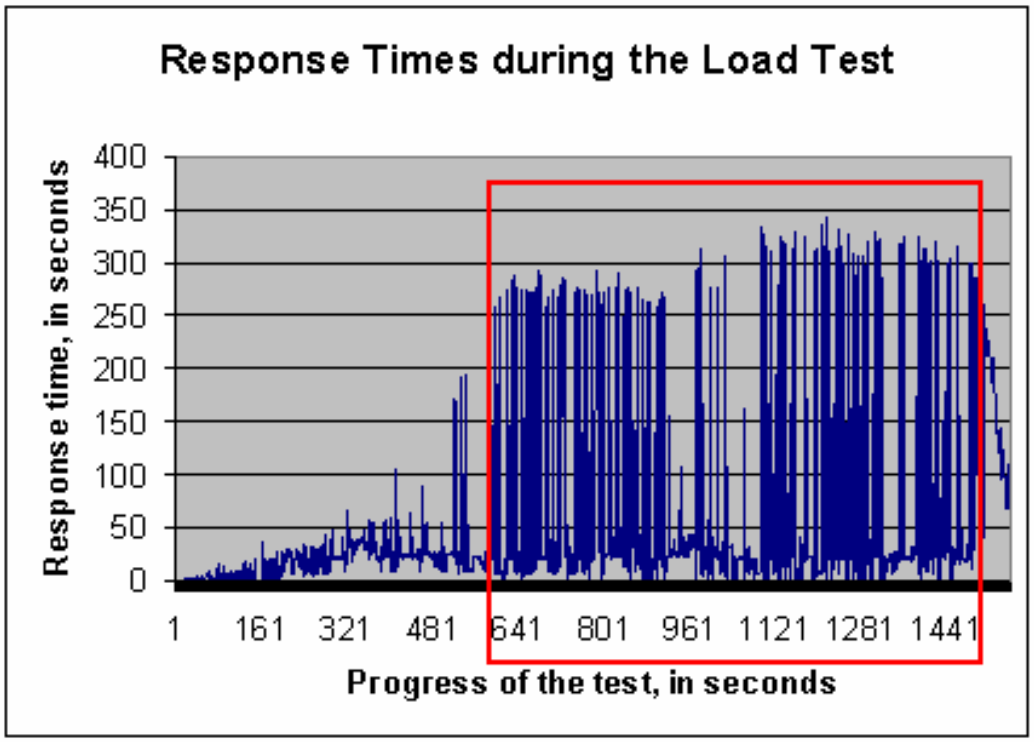

Figure 13: Sample response times during a load test

The execution should be performed till the moment, when the response times (or another major metric used) do not vary very much (when its behavior is clear and predictable). The figure below displays the change of the average response time and individual response times during the conduction of a test. The test should be performed until the line representing the metric used becomes pretty much horizontal, which means it does not change much.

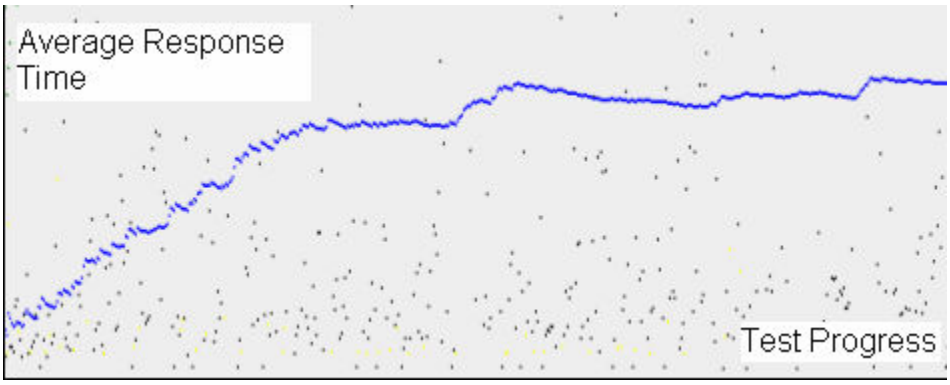

Figure 14: Average response time during a sample load test

The following figures represent the progress of a load test performed by WebUnit, which, as mentioned earlier, executes a test via a WEB browser and can output data about the progress of a test into the standard output, log it or send it somewhere else.

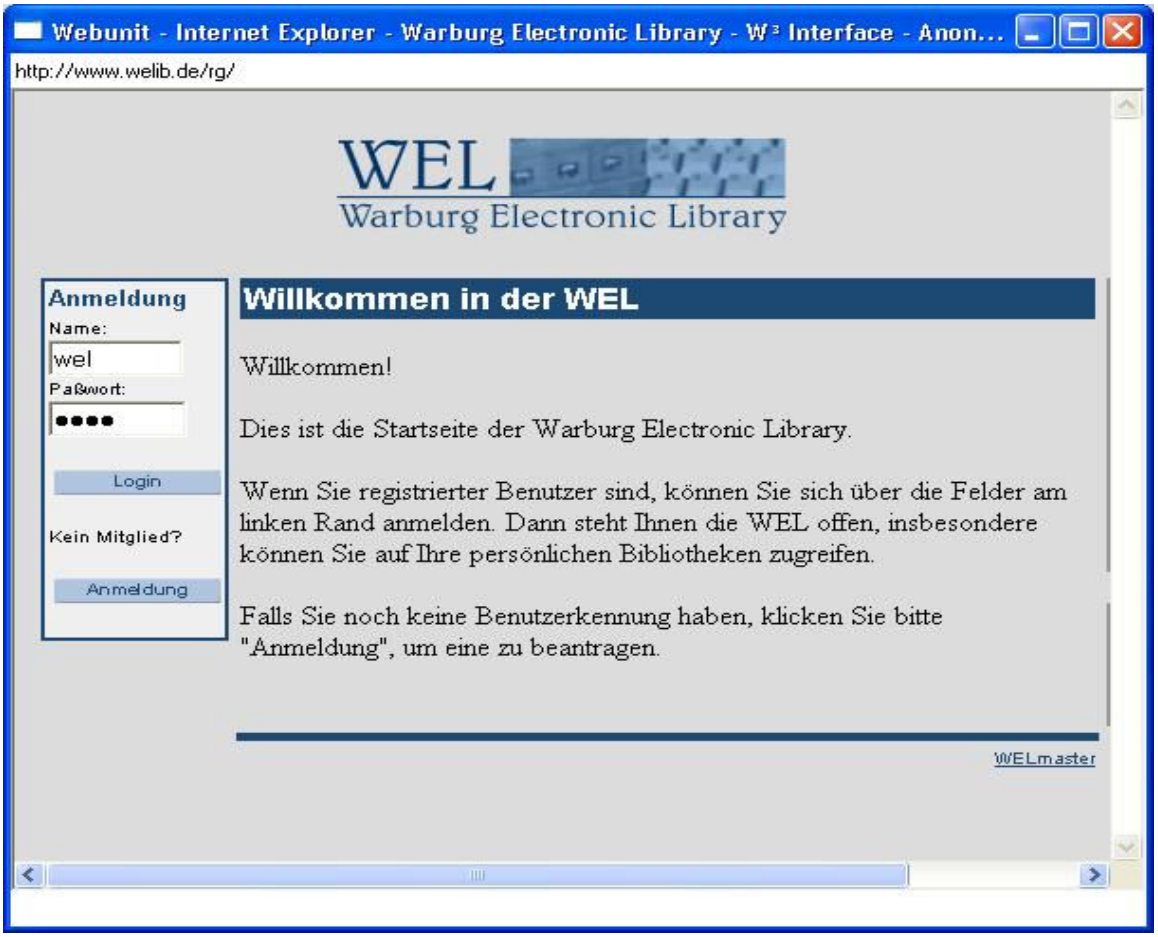

Figure 15: A WebUnit driven WEB browser is logging in into WEL

The result outputted by the WebUnit's script (listed in the appendixes) during the execution is displayed below.

```
Server:Starting Server--
Server:Starting Server--
Server:Starting Server--
Show Start Page [thread: 0]          1833    ms.
Current Avg Delay                    1833    ms.
Show Start Page [thread: 1]          1141    ms.
Current Avg Delay                    1487    ms.
Show Start Page [thread: 2]          2003    ms.
Current Avg Delay                    1659    ms.

                                     …

Go to Login Page [thread: 1]         3445    ms.
```

Show ClassifierIndex [thread: 2]	2744	ms.
Current Avg Delay	4328	ms.
Show a Classifier [thread: 0]	4546	ms.
Current Avg Delay	4342	ms.
Show a Classifier [thread: 1]	4526	ms.
Current Avg Delay	4353	ms.
Show a Classifier [thread: 2]	3595	ms.
Current Avg Delay	**4311**	**ms.**
Server:End Server Normal		
Server:End Server Normal		
Server:End Server Normal		

Table 4: Parsed result of a load test execution by WebUnit

The person in charge of the execution of a test should check the value of "Current Avg Delay" and stop the test when the value does not vary much for a certain period of time.

5.3 Analysis of the Results

First of all, it must be made clear whether a load test has been successful, which means that the results of the test are correct and could be used to make judgments. It must be checked whether the network has been a limiting factor, whether the measured response times are fluctuating insignificantly, whether the system performed as had been expected and so on.

The results of the load test have to be thoroughly analyzed. One of the first things to do is to check, whether there were cases when the server crashed (in some way or another) or there were other anomalies under a significant load. The reasons have to be identified (what could cause this purely depends on the internal structure of the system. A reason could be, for instance, that the database queries timeout).

Another very important aspect is to identify, what exactly slows the system down most significantly, where are the current bottle necks. This can be done by checking, which pages take more time to load on average. This is particularly bad, if the bottle neck is the functionality which is used often enough. Sometimes, the black box testing might be not enough to identify the problem. Thus, the load test can be conducted again and the logging should be implemented in various parts of the code. The same load test's configuration might be used. This could result e.g. in finding out that there is one method, which is too slow, and all the pages, which use it, are therefore affected.

Particular attention should be driven to the way, how the resources used are released, when there are fewer users. Ideally, the resources have to be released very fast, after the processing of a request is finished.

If there are no major pitfalls identified, the greatest value from the conduction of a load test is the answer to the question, how many concurrent users can the software system serve within a desired response time. This information can be used by different parties, allowing for planning the investments into the platform, predict its behavior, find the parts of the system which are most resources consuming and so on.

Load testing tools normally provide some means for representing the results of a test, like constructing corresponding graphs, aggregating the data, etc. That enables to make first conclusions quickly just by looking at the graphs.

5.4 Case Study: "Warburg Electronic Library"

The load testing of the Warburg Electronic Library has been performed, using the configuration described in the earlier chapters, with different numbers of concurrent users: 5, 10, 20, 30, 40, 50, 75 and 100, acting according to the scenarios described in the chapter 3. A single computer has been used to perform the test. The computer has been placed close to the server regarding the network topology, but not in the local network of the WEL server. During the test it has been monitored that neither network's connection nor test-computer's resources became a bottleneck. The load test has been performed on Friday evening when no significant usage of the service is expected. The figures below represents the results derived.

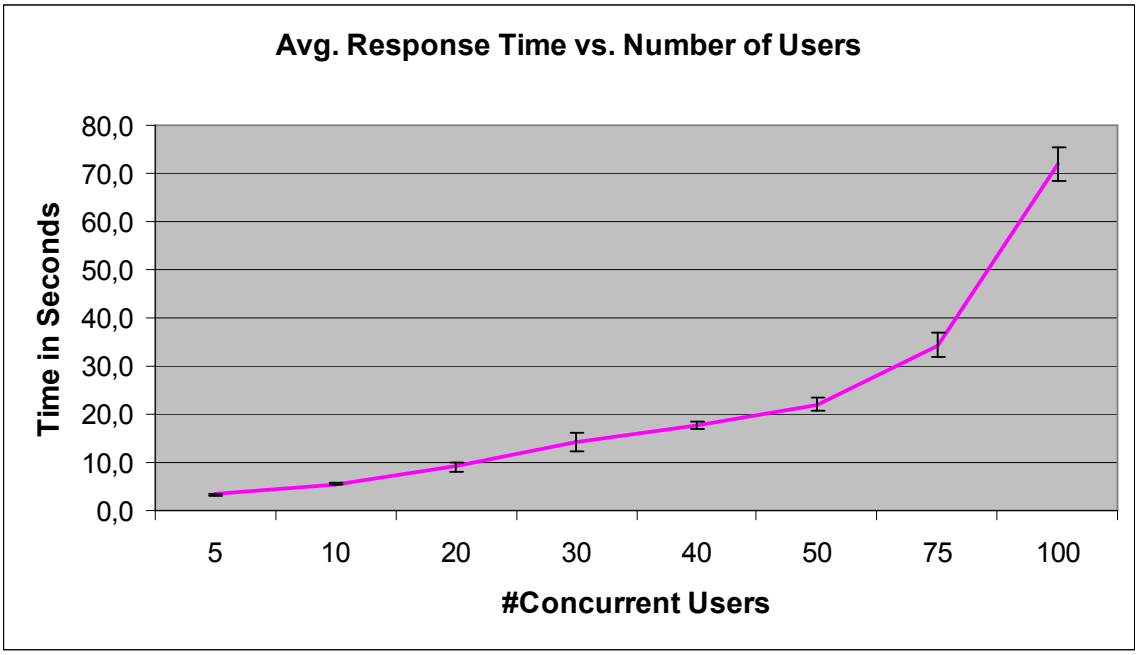

Figure 16: Dependency of the average response time on the number of concurrent users

As can be seen from the figure 16 the increase in the average response time has rather linear dependency until the delay becomes too high on a number of concurrent users (The average response time approximates to the number of concurrent users divided by 2). The figure also displays error bars, which represent standard deviation. The error level is acceptable, which shows that the measurements have been taken when the ramp up phase has already finished and the average delay did not fluctuate much. If we assume that the delay which is not much disturbing for users, must be less than 5 seconds, then the current system implementation (involving both hardware and software) can only afford to have about 9-10 concurrent users. Whether it is a lot or not, generally depends on the type of the application, its popularity, the desired quality of the service provided, etc. The WEL has about 260 registered users. Most of them use the system occasionally,

spending very few (2-3 on average) minutes browsing, so the probability of having more than 9-10 concurrent users is rather low.
From the user's perspective the average delay between page-views, is normally short, but the values of the maximal delay which user might experience are significantly higher.

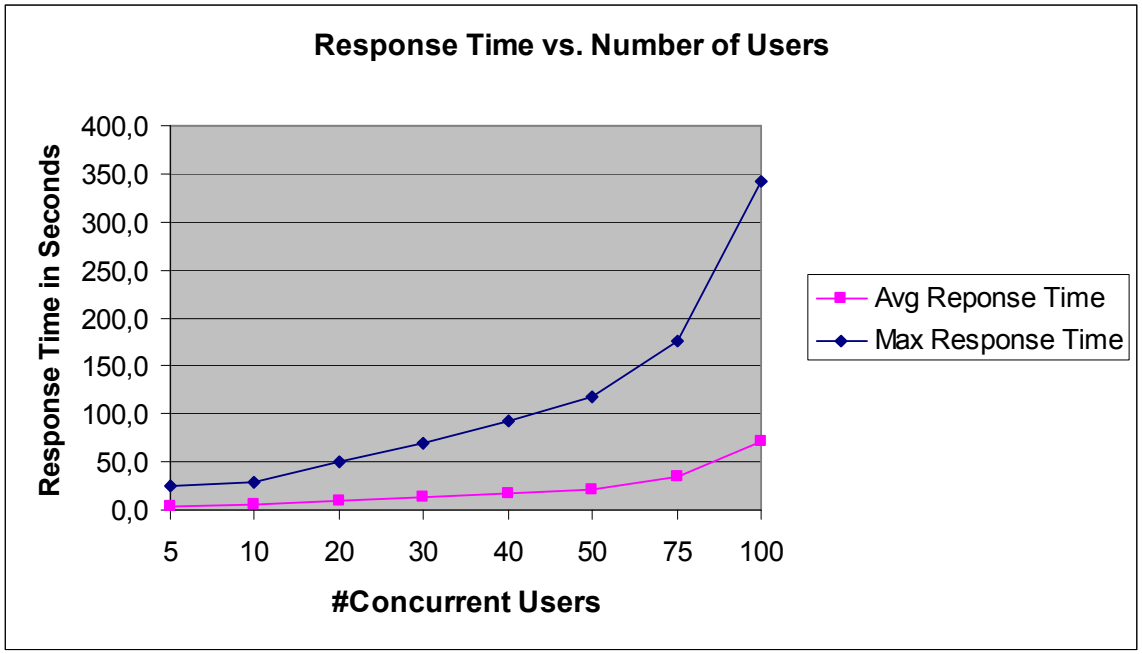

Figure 17: Dependencies of the maximal and the average response times on the number of concurrent users

The figure above displays both average, and the maximal delays between page-views. It appears that the maximal delay grows much more significantly when the load increases. Moreover, even with 5 concurrent users (which is quite probable situation) the maximal delay reaches, as has been derived from the load test, 25 seconds! This means that there is certain functionality which is simply slow regardless the load.

It also shows that there are pages of different complexity and structure. Some WEL objects are associates with many categories, whereas others are too specific and the retrieval of the pages associated with them does not take too much time. The graph below gives an idea of the variation of the response times when the server is being tested with 40 simultaneous users. As can be seen from the figure, individual response times vary very much. The objects which have been accessed during the load tests have been taken from the earlier generated data files. The duration of the tests has been different, because e.g. the ramp up phase for 100 concurrent users has been made to last longer than for 5 users. The part of the objects accessed in the beginning has been the same. If some test has been running longer, of course, it has been accessing other objects as well, which have not been used for other tests. But the behavior of the *average* response time of the server has been quite stable after the ramp up phase in all of the tests.

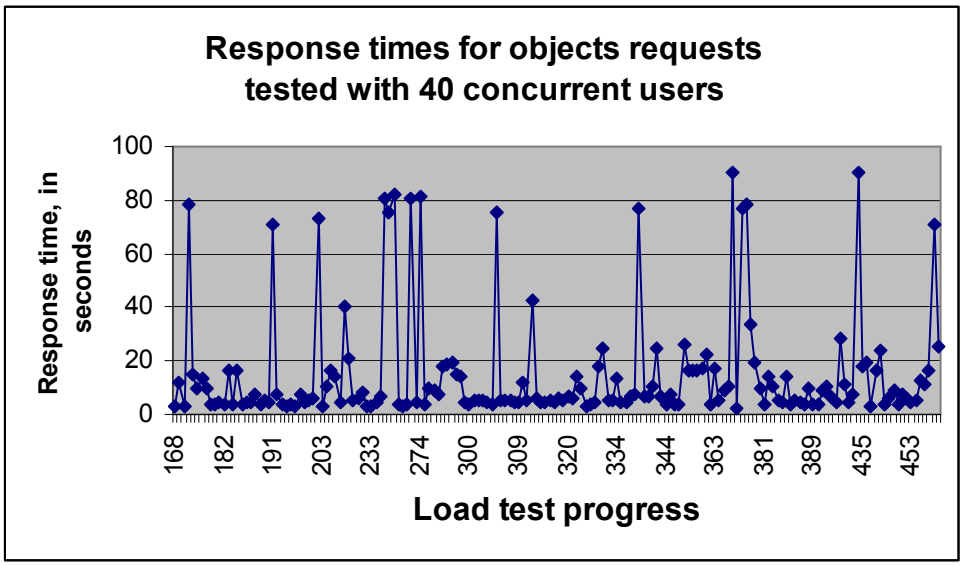

Figure 18: Response time for objects' requests tested with 40 concurrent users

The next table reveals the data used to construct the figures above.

Concurrent Users	Average Response Time	Max Response Time	Standard Deviation of the Avg. Response Time
5	3,30	25,30	0,085
10	5,50	29,20	0,162
20	9,14	50,60	0,894
30	14,10	70,10	1,948
40	17,70	92,30	0,725
50	22,01	117,90	1,386
75	34,40	176,60	2,502
100	71,90	342,50	3,328

Table 5: The results of the load test

Let's look at the operations / pages, which are the slowest. As could be foreseen, the order does not change with the increasing load. The following figures represent the numerical values for the load test with 40 concurrent users.

Page Type	Average Response Time (in seconds)
Search	73,8
Object request by user from cluster 2.	22,4
Object request by user from cluster 1.	14,9
Log in	5,7
Start page	0,7
Log out	0,3

Table 6: Average response time per page type

The following figure visualizes the data from the table above.

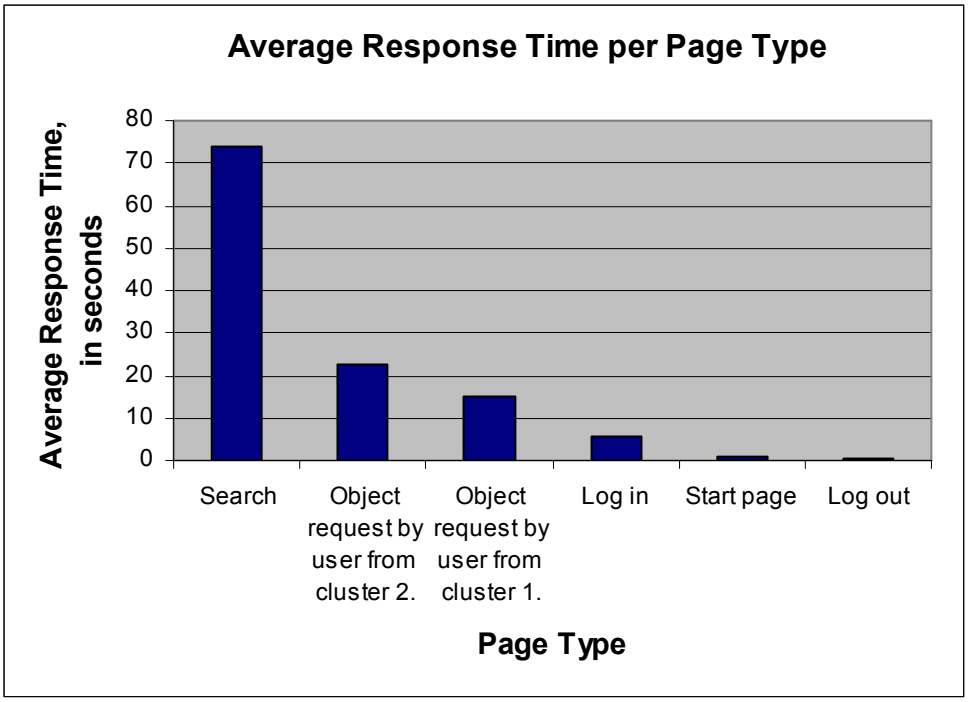

Figure 19: Average response time per page type

The load test performed has been purely black box testing. So, the internal implementation of the system to the large extend is unknown. Nevertheless, analyzing the results, some decent assumptions can be made. Let's have a look at each of the pages and try to surmise the reasons explaining the results derived:

- The "Log out" page and the "Start page" apparently do not need to execute any database queries and are very fast even when the load is high.
- The "Log in" page needs to execute at least one database query to check whether the password for the username entered is correct. And probably some more to apply corresponding roles and rights system for the user. The table holding the users' data is very small and presumably is indexed properly. The average delay exceeds 5 seconds, which means that even very simple database queries are very slow when the load is significant. This might be due to the database's configuration having a limited queries' throughput.
- The pages displaying requested objects to the users from both clusters are naturally slower than the "Log in" page, because the data is being taken from the larger tables, access policies are to be checked, and the links to the related objects have to be displayed, which requires making multiple database queries.
- The "Search" page is currently the slowest. The maximal response times are always contributed by the requests of the search pages. This page is too slow even when the load on the system is minimal. This might be quite easily explained. The search page looks for all Classifier, which names include a certain keyword. This means the database query must run over a table with several thousand rows, and check each (presumably) not indexed field with the LIKE-type queries. Moreover, the search page counts all the matching rows, and (by default) displays the list of the first 20 objects, with their related links. It means that if the keyword is included into the names of at least 20 classifiers (e.g. "Präsident"), the number of "logical queries" which need to be executed might be at least 60. Of course, the real number of the queries depends on their complexity, database structure and the way to match data in different tables, but deriving the data about 20 classifies is expensive from the perspective of the resources consumption.

It is known, that every 2nd user from the cluster representing "users looking for something concrete" is using the search functionality once within his/her session. So, even with the small load, that kind of user might need to spend around 15-20 seconds waiting for the results of the search page.

Overall, considering the current number of users using the system, the performance of the system might be sufficient, except for the very slow and heavy search page. The search page could be changed to contain fewer references to the related objects. Since most users use the default settings, the default behavior could also be changed to include less functionality.

6. Summary and Future Work

This chapter gives a brief summary about the both theoretical and practical parts of work, as well as a view on a future work.

6.1 Summary

The goal of this study was to analyze the techniques for the performance improvement of a web-based software system by analyzing its users' behavior and performing an adequate load test. The study described different ways to collect and interpret the information about the system's users, which is essential for the load test.

Two Java programs have been developed for this purpose within the practical part of the work: the first one to parse the available WEL log files and identify browsing behavior of the users ("LogFilesAnalyser"), and the second one ("ClusterGenerator") to perform cluster analysis over the collected data. Arranging users into 6 clusters allowed for judgments about different types of users and the functionalities of the system they use more frequently. Four clusters have been intentionally left out from the further stages of preparation to the load test, because they represented too small groups of users (around 6% in total), who's behavior did not affect the performance of the system much. The behavior of the users from 2 major clusters has been analyzed more thoroughly. It appeared to be that users from one of the clusters tended to look for something concrete, while the users from another cluster preferred to browse around the system mainly.

Further, all conceptual stages of a load test have been analyzed. A number of load testing tools has been reviewed and evaluated. A free open source load testing tool JMeter has been chosen for the needs of the load test of the Warburg Electronic Library system. This tool provided simple means to meet WEL load test requirements. This is mainly related to the ability of using big amounts of variable data, which was needed to simulate WEL users accessing different types of objects with different degree of their interrelation's complexity. To generate corresponding "browsing paths" for simulated users, another Java program ("TestPathGenerator") has been developed.

After that an adequate load test has been designed, configured and executed. The results have revealed the performance of the Warburg Electronic Library system under different loads (different number of concurrent users). The respective dependency between a concurrent number of users and the average server's response time has been identified.

The following figure summarizes the practical part of the study, showing the sequence of the interactions between different constituents involved into the performance of the Warburg Electronic Library's load test.

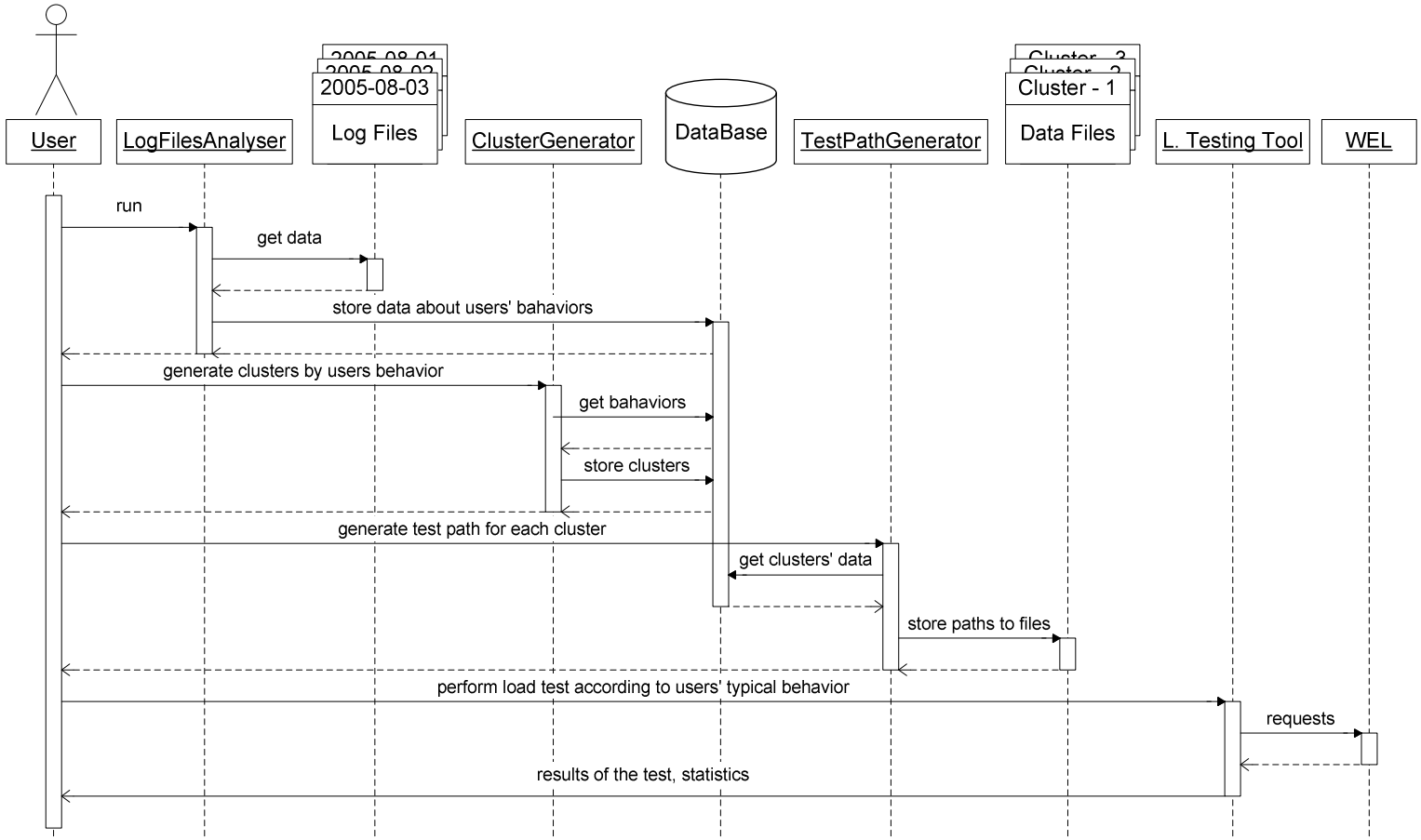

Figure 20: Sequence diagram depicting interaction between different constituents required for the load test performance.

6.2 Future Work

The load test serves for *measuring* performance. The executed load test has revealed several directions for potential performance improvement. A performance test might be developed from the current load test in future, allowing for the evaluation of the effectiveness of the system's improvements and the correctness of the assumptions made in this study regarding the bottlenecks.

To perform the load test on the WEL system, several loosely integrated "components" have been used. This is suitable because all components can be replaced by other ones if needed. But this also introduces some limitations, requiring some manual work to be done to adopt data from one interface to another. In case load tests are to be performed repeatedly, better integration of the components might be desirable. That would decrease the time needed for one full load test cycle. This is not an easy task since each software project has own unique requirements and limitations. Nevertheless, some steps for performing an adequate load test, which would only have system usage logs and a minimal human intervention as an input could be done.

The software programs and approaches developed in this study perfectly suit the needs of the Warburg Electronic Library's load test. However, the WEL system is quite specific in some aspects. For instance, all the pages (except very few) are referred by a unique object id. The log files only contain a sequence of object ids user accesses at specific moments of time. The structure of most software systems with web presentational layer is definitely different (the log files normally contain various URLs with various number and character of parameters accesses by users). The extensions to the developed programs supporting different types of logs, clustering algorithms, and load testing tools might be added. That would make the approaches used in this study more general and more widely applicable for testing other software projects.

The load testing tool JMeter has been an optimal choice for the needs of WEL load test, but it lacks a convenient automated approach for measuring system performance with different number of concurrent users in the system. JMeter, being a free open source software, could be enhanced to be more convenient for frequent use.

Appendices

Appendix 1: Sample WebUnit program load-testing the WEL

Class SimpleLoadTest

```
/**
 *      This simple program shows how the load test can be conceptually performed with
 *      WebUnit.
 */
package project_work.load_test;

import com.zeborg.webunit.*;
import com.zeborg.webunit.util.*;
import java.io.*;


/** This simple example opens WEL site, logs in, and navigates around a little.*/
public class SimpleLoadTest
{
 static final int NUMBER_OF_CONCURRENT_USERS = 3;
 static final String login_username = "dummy";
 static final String login_psw = "dummy";
 static final String URL = "http://www.welib.de";

 public static void main(String[] args)
 {
  // initialize; set properties
  String webEnvVarName ="zeborg.webunit.msvm.cp";
  if (System.getProperty(webEnvVarName) == null)
  {
   File curDir = new File(".");
   File libDir = new File(curDir, "../lib");

   File rmijar = new File(libDir, "rmi.zip");
   File msJar = new File(curDir,"../webunit-ms.jar");
   String val = rmijar.getAbsolutePath() +";" + msJar.getAbsolutePath();
   System.setProperty(webEnvVarName,val);
  }

  // launch 'NUMBER_OF_CONCURRENT_USERS' threads
  WELThread[] threads = new WELThread[NUMBER_OF_CONCURRENT_USERS];
  for (int i = 0; i < NUMBER_OF_CONCURRENT_USERS; i++){
        threads[i] = new WELThread(URL, login_username, login_psw);
        }
```

```
  }
}
```

Class WELThread

```
package project_work.load_test;

import com.zeborg.webunit.*;
import com.zeborg.webunit.util.*;
import java.io.*;
import java.lang.Thread;

/**
 *        This class is a Thread for accessing WEL start page, logging in,
 *  and browsing around a little bit.
 */
public class WELThread implements Runnable
{
  static int threads_number = 0;
  static long total_delay = 0;
  static long page_views = 0;
  String URL, login_username, login_psw;

  long previous_timestamp = 0;
  int this_thread_number = 0;

  // Constructor, sets few properties to access WEL
  RequestThread(String URL, String login_username, String login_psw){
          this.URL = URL;
          this.login_username = login_username;
          this.login_psw = login_psw;

          this_thread_number = threads_number;
          threads_number++;
          Thread t = new Thread(this);
          t.start();
  }

  // Thread's executable method
  public void run()
  {
    try
    {
     WebClient wc = null;
     try
     {
```

```
                // the following line will initialize the IE browser
                wc = DefaultWebFactory.getFactory().newWebClient();

                // request the WEL start page
                register_pageview("Start");
                wc.openPage(URL);
                WebResponse currentPage = wc.getCurrentState();
                // measure the response time
                register_pageview("Show Start Page");

                // find the log-in link (the image 'zugang.gif') & click it!
                currentPage.getLink("zugang.gif", WebTag.FIND_CONTAINS).click();
                currentPage = wc.getCurrentState();
                register_pageview("Go to Login Page");

                // now log in!
        WebTextField username = (WebTextField) currentPage.getRootTag() .
findFirst("name", "user", WebTag.FIND_EQUALS, WebTextField.class);
        username.setValue(login_username);

        WebTextField password = (WebTextField) currentPage.getRootTag() .
findFirst("name", "password", WebTag.FIND_EQUALS, WebTextField.class);
        password.setValue(login_psw);

        // normally we would click a submit button, but as WEL does not have it,
        // contruct a GET request.
        WebForm form = currentPage.getFormWithName("loginform");
        String form_action = form.getAttribute("action").getValue();
                String form_submit_url = URL + form_action + "?user="
                        + login_username + "&password=" + login_psw;
                Thread.currentThread().sleep(5000);

                // "submit" log in page
                wc.openPage(form_submit_url);
                // get the page
                currentPage = wc.getCurrentState();
                register_pageview("Perform Login");

                // go to the list of libraries (categories)
                currentPage.getLink("img/biblio.gif",
                                WebTag.FIND_CONTAINS).click();
                currentPage = wc.getCurrentState();
                register_pageview("List Libraries");
                Thread.currentThread().sleep(2000);

                // go to the list of libraries (categories)
```

```
                currentPage.getLink("September", WebTag.FIND_CONTAINS).click();
                currentPage = wc.getCurrentState();
                register_pageview("Show ClassifierIndex");
                Thread.currentThread().sleep(3000);

                currentPage.getLink("Afghanistan", WebTag.FIND_CONTAINS).click();
                currentPage = wc.getCurrentState();
                register_pageview("Show a Classifier");

                // Sleeping 10s so that the browser window could be looked at.
                Thread.currentThread().sleep(10000);
    }
    finally
    {
     if (wc!=null) wc.close();
    }
   }
   catch(Exception e)
    {
      e.printStackTrace(System.out);
    }
  }

        /**
         *      Outputs how much time has elapsed from the last call of this method
         *      @return (long) delay from the last method's call
         */
   public long register_pageview(String sEvent_name) {
        long delay = 0;
                if (previous_timestamp > 1){
                page_views ++ ;
                        delay = System.currentTimeMillis() - previous_timestamp;
                        total_delay += delay;
                        // display who much it took to retrieve page 'sEvent_name'
                        System.out.println(sEvent_name + " [thread: "
                                + this_thread_number +"];" + delay + "; ms.");
                        // display the current average.
                        System.out.println("Current Avg Delay ;"
                                + (total_delay / page_views) + "; ms.");
                }
                previous_timestamp = System.currentTimeMillis();

                return delay;
   }
}
```

Appendix 2: Detailed Data about WEL User Clusters

Cluster ID	1	2	3	4	5	6
Description	Users looking for something concrete	Browsing users	neglig.	neglig.	neglig.	neglig.
percentage of users	54,75	39,16	2,28	1,52	1,52	0,76
percentage of objects accessed	37,49	53,3	1,25	5,72	0,91	1,33
percentage of usage of relations	35,11	54,12	0,9	7,43	0,93	1,52
percentage of going back	25,18	67,69	0,71	4,67	0,64	1,11
total users	144	103	6	4	4	2
total requests	6561	9329	219	1002	159	233
avg number of sessions	7	7	5	3	1	2
avg session objects accessed	7	12	7	78	40	58
avg session follow links	5	8	4	72	29	48
avg session usage of search	1	0	2	1	0	1
avg session usage of search for card	0	0	1	0	0	0
avg session usage of card chain	0	0	0	2	0	1
avg session going back	1	3	2	21	11	18
avg session reaccessing card	0	0	0	2	0	0
avg session reaccessing classifier	1	1	1	14	4	11
avg session access of classifierIndex	1	3	2	5	10	9
avg session access of card	1	1	2	30	1	5
avg total objects accessed	45,56	90,57	36,5	250,5	39,75	116,5
avg usage of relations	30,77	66,31	18,83	234,5	29,25	96
avg usage of search	5,14	3,6	10	2,75	0,25	3

avg objects not identified	5,47	24,85	1	1,5	2	11,5
avg search a Card	1,79	1,5	5,67	0	0	0
avg search classifier	3,02	1,9	2,33	2,5	0,25	3
avg search hierarchicalExtent	0,33	0,2	2	0,25	0	0
avg usage of specialization	9,47	28,49	2,83	39	10,75	40,5
avg usage of classification	5,84	5,14	3,67	73,75	1,25	11
avg usage of classification fromPic	2,73	1,5	1,5	34,75	0	4
avg usage of a Card chain	0,45	0,88	0,5	6,5	0	2
avg usage of classification hierarchical	1,9	4,4	1	29,5	1,75	0
avg usage of affiliation	9,73	25,11	8,33	24,5	15,5	38,5
avg usage of aggregation	0,49	0,57	1	26,5	0	0
avg usage of aggregation reverse	0,16	0,23	0	0	0	0
avg access of classifierIndex	9,65	20,66	7,67	13,25	10,25	17,5
avg access of classifier	25,4	57,87	15,5	107,3	26,5	88
avg access of hierarchicalExtent	2,39	4,83	3	29,75	1,75	0
avg access of a Card	8,13	7,2	10,33	100,3	1,25	11
avg going back	11,8	44,35	8	78,75	10,75	37,5
avg reaccessing classifierindex	3,94	13,42	2,67	8,5	6,75	14
avg reaccessing classifier	6,35	26,67	3,83	54,25	4	23,5
avg reaccessing hierarchicalExtent	0,56	2,61	0,83	6,5	0	0
avg reaccessing a Card	0,08	0,04	0,07	2,16	0	0

- Java code to group users into clusters
- SQL queries to manipulate data from cluster analysis(?)

References

1. Apache web site http://httpd.apache.org/docs/1.3/logs.html#accesslog http://httpd.apache.org/docs/1.3/mod/mod_log_config.html
2. Cheng, Yu-Chung, Hoelzle, Cardwell, Savage, Voelker: Monkey See, Monkey Do: A Tool for TCP Tracing and Replaying. University of California, USA, 2004
3. Cluster analysis http://en.wikipedia.org/wiki/Data_clustering
4. Cluster analysis http://www.statsoft.com/textbook/stcluan.html
5. Definitions and explanations of different types of server tests http://www.keylabs.com/faq.shtml
6. Elbaum, S., Karre, S., Rothermel, G.: Improving Web Application Testing with User Session Data. Oregon State University Corvallis & University of Nebraska - Lincoln, USA, 2003
7. Hendrickson, Elisabeth, Aveo Inc: Stress testing load on a server. International Conference On Software Testing Analysis & Review, CA, USA, 2000
8. How to stress or load test http://wiki.rubyonrails.org/rails/pages/HowToStressOrLoadTest/versions/1
9. Mobasher, B., Cooley, R., Srivastava, J.: Creating Adaptive Web Sites Through Usage-Based Clustering of URLs, University of Minnesota, Minneapolis, USA
10. Mobasher, Dai, Luo, Sun, Zhu: Combining Web Usage and Content Mining for More Effective Personalization, Illinois, USA, 2000
11. Montgomery, C. Douglas, Runger C. George Applied Statistics and probability for engineers III edition, 2002
12. Multivariate statistics http://folk.uio.no/ohammer/past/multivar.html
13. Netcraft's "February 2006 Web Server Survey" http://news.netcraft.com/archives/web_server_survey.html
14. Sandino, R., Giachetti, R.: Using Simulation Modeling to Predict Scalability of an E-commerce Website. Florida International University, USA, 2001
15. Talagala, Nisha, Asami, Satoshi, Patterson, David: Usage Patterns of a Web-Based Image Collection, University of California at Berkeley, USA.
16. Testing tool Alertsite http://www.alertsite.com/
17. Testing tool FunkLoad http://funkload.nuxeo.org/
18. Testing tool Httperf http://www.hpl.hp.com/research/linux/httperf/
19. Testing tool JMeter http://jakarta.apache.org/jmeter/
20. Testing tool Siege http://joedog.org/siege/
21. Testing tool SiteStress http://www.webmetrics.com/loadtesting.html
22. Testing tool Watir http://wtr.rubyforge.org/
23. Testing tool Webserver Stress Tool http://www.paessler.com/webstress/

24. Testing tool WebUnit http://webunit.sourceforge.net/
25. UML Series http://www.developer.com/design/article.php/2206791
26. Variance information http://en.wikipedia.org/wiki/Variance
27. Ward's linkage http://www.statistics.com/content/glossary/w/wardslnkg.php
28. Web site test tools and site management tools
    http://www.softwareqatest.com/qatweb1.html
29. White box vs. Black box testing
    http://www-128.ibm.com/developerworks/rational/library/1147.html

Declaration

I declare within the meaning of the examination and study regulations of the international master program course Information and Media Technologies: this project report has been completed by myself independently without outside help.

City	Date	Signature
_____	_____	_____