FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

Repackaging Web Pages using Advanced User Behaviour Analysis

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Business Informatics

eingereicht von

Maximilian Aster
Matrikelnummer 0825493

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Univ.Prof. Mag.rer.nat. Dr.techn. Reinhard Pichler
Mitwirkung: Dipl.-Ing. Ruslan Fayzrakhmanov

Wien, 25.07.2013

(Unterschrift Verfasser) (Unterschrift Betreuung)

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Repackaging Web Pages using Advanced User Behaviour Analysis

MASTER'S THESIS

submitted in partial fulfilment of the requirements for the degree of

Diplom-Ingenieur

in

Business Informatics

by

Maximilian Aster
Registration Number 0825493

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Univ.Prof. Mag.rer.nat. Dr.techn. Reinhard Pichler
Assistance: Dipl.-Ing. Ruslan Fayzrakhmanov

Vienna, 25.07.2013

(Signature of Author) (Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum)	(Unterschrift Verfasser)

Kurzfassung

Der Fokus am Markt der internetfähigen Endgeräte verschiebt sich, aktuellen Trends folgend, immer mehr von Desktop-Geräten hin zu mobilen Geräten wie Tablets oder Smartphones. Dies hat zur Folge, dass sich auch Webseiten an diese Begebenheit anpassen müssen. *Advanced User Behaviour Analysis* versucht die Effektivität und Benutzbarkeit von Webseiten im Allgemeinen, aber auch speziell für ausgewählte Benutzergruppen zu verbessern. Dazu werden Informationen, die aus der Aufzeichnung von Benutzerinteraktionen mittels mobiler Geräte gewonnen wurden, herangezogen. Der Begriff soll außerem klar machen, dass neue Herangehensweisen evaluiert und verwendet werden um dieses Ziel zu erreichen. Es gibt allerdings noch weitere Möglichkeiten diese Daten zu verwenden. Nämlich um Webseiten oder den ihnen zu Grunde liegenden Geschäftsprozess automatisch in eine für eine andere Benutzergruppe ausgelegte Sicht zu transformieren.

Um dies zu erreichen wird ein allgemeines Prozessmodell vorgestellt, dessen einzelne Phasen durchlaufen werden müssen um eine Webseite zu transformieren. Es stellt auch die Basis für die als Teil dieser Arbeit entwickelten Lösungen dar. Des Weiteren berücksichtigt es den neuesten Stand der Kenntniss und Technik im Bereich Web Science. Die implementierten Artefakte zeigen wie eine dazugehörige Software Architektur aussieht und welche Möglichkeiten sich dadurch ergeben. Ein wichtiger Aspekt ist wie die gesammelten Daten über Benutzerinteraktionen verwendet werden um die Qualität der Transformation, dadurch dass nur Inhalte ausgewählt werden die für andere Benutzer relevant waren, zu verbessern.

Abstract

Current trends in the market of Web-enabled devices are moving the focus from desktop web pages to pages optimised for a set of other devices like smartphones or tablets. *Advanced User Behaviour Analysis* tries to improve the effectiveness and usability of web pages in general and for specific user groups, using information that is gained from tracking users utilizing those mobile devices. The term points out the fact that new approaches are evaluated and used in order to achieve this goal. But the user behaviour data collected this way can also be used to adapt web pages and automatically transform pages and even the web applications logic flow into a new kind of representation, specifically for a certain target group.

Therefore a general process is defined to describe the various phases that have to be gone through to transform or repackage a website. It serves as the basis for the solutions, which were built as part of this thesis, and incorporates state of the art concepts and methods from various fields of Web Science. The implemented artefacts demonstrate how an appropriate architecture looks like and what additional possibilities open up. In the end the collected data about user behaviour is employed to optimise the transformation of the web pages by selecting only content which showed to be important to other users.

Acknowledgements

This master thesis has opened me the possibility to explore interesting topics and to work with a lot of great people.
I would especially like to thank Robert Baumgartner, without whose support it would not have been possible. Secondly I would like to thank my supervisor Reinhard Pichler for all the administrative help and for giving feedback.

A big thank you, for the productive and enjoyable teamwork, goes out to my project team members David Neukam and Sebastian Morgenbesser.
I would also like to thank the staff of the DBAI group at the Vienna University of Technology, especially Ruslan Fayzrakhmanov, Christoph Herzog, and Bernhard Krüpl–Sypien, for the discussions which always resulted in valuable ideas.

Finally, my deepest gratitude goes to everybody close to me who has given me support and encouragement.

Contents

CHAPTER 1

Introduction

1.1 Motivation

The ultimate motivation for this project and the resulting theses is the increasing domination of smartphones in the World Wide Web these days. The rising capabilities of these modern devices enable their user base a lot of functions and possibilities to perform every day actions, such as booking flights, reading the news, looking up facts, even doing work and many more things on the web. Despite the possibilities users have got using their mobile devices, many web pages or web applications are not optimised for them. Of course lots of mobile websites have evolved these days, but many of them lack usability and do not provide important functions or information. A solution has to be found in order to increase the capabilities and usability of websites and web applications towards mobile devices.

How do website owners find out if their mobile appearances on the web are adopted by the users and used in a meaningful way? How do they know what could be improved? Of course one could make surveys or use statistic evaluations in order to find out how many people are using the mobile site and how happy they are with it. A much better solution would be to directly analyse the surfing behaviour of actual users that are visiting the particular sites. Many tools are available by now to track users on websites and to evaluate all kinds of data. The main problem all these tools are facing is that current approaches to track user behaviour in order to improve web pages are mainly focused on desktop users and do not consider the special group of mobile users directly. Why not focus on analysing the specific user group the website should be improved for? This is exactly what the idea of *Mobile Viewport Tracking* is about. It tries to improve the effectiveness and usability of web pages in general and for specific user groups, using information that is gained from tracking users utilizing mobile devices. The term *Advanced User Behaviour Analysis* points out the fact that new approaches are evaluated and used in order to achieve this goal. The main focus is to use mobile devices to track typical user interactions that can be archived and further interpreted, in order to analyse or adapt web pages.

The first step of a practical implementation is to track position data within a web page, and visually represent the collected data in the form of a so called heatmap. The next step is to take the

basic concept and develop a more sophisticated approach which uses the collected information in a broader view in order to enable semi-automatic or automatic analysis, repackaging of websites, and also archiving and visualising user surf sessions in a meaningful way. These tasks contain a lot of challenges that will be dealt with in this project and the resulting theses. The information that can be retrieved from tracking users is limited and has to be interpreted somehow in order to optimise web pages. This can either be done manually, using some kind of reporting tool, or automatically by a program which could be very complicated. For an automatic interpretation, the tracking system would have to be able to recognise patterns and similarities in the surfing behaviour of users and to derive conclusions from it. Another challenge is the repackaging of web pages where important parts have to be identified and rearranged to build an optimised version of the page. Archiving and visualising or replaying user interactions, requires tracking of all types of actions a user performs during a surf session, such as entering text, clicking links, hopping from one site to another, etc.

1.2 Problem statement

A big problem of the Web in its current state is that web pages are designed for desktop browsers and that it is expensive to develop specific solutions for different target groups, e.g. for mobile devices. Every representation different from the original comes with certain requirements, which are often based on constraints, that have to be fulfilled. For example the smaller size of mobile devices or the fact that request and response times are usually slower compared to those of desktop machines [46].

This project resp. thesis will tackle the problem by evaluating an automatic approach to adapt or optimize a web page for specific target devices. This task will further be called repackaging. The basic idea is to improve and optimize a web page, either by providing improvement suggestions, e.g. by highlighting certain content, or by transforming the content into another representation. As a concrete use case smartphones are chosen as target devices for the repackaging.

During the master thesis, the following research questions will be answered:

- How can repackaging of a web page for a specific target device be achieved?
- How could a feasible model / process look like?
- How could an appropriate architecture look like?
- How can the content used for repackaging be determined?
- How can the output of user behaviour analysis be used to support it?
- Can the captured web content be modelled and further processed?

1.3 Aim of the work

Following the chosen methodology artefacts will be developed and evaluated to answer the stated research questions. This requires the creation of a theoretical model or process describing how repackaging can be achieved and what the main parts or steps will be. This will include the evaluation of methods and concepts from different Web Science fields, e.g. to extract content from web pages, represent and remodel it, and transform it.

Besides that, two practical solution will be implemented. The first one has to be capable of injecting scripts into a web page to highlight the content which was identified as relevant using *Advanced User Behaviour Analysis*. The second solution will allow to create a completely repackaged version of a chosen web page. This will be performed in a server-side environment and will make use of the previously identified concepts of the theoretical part.

For the evaluation of the design descriptive and experimental methods, i.e. controlled experiments and scenarios, will be used to demonstrate the utility of the implemented artefacts and to answer the defined research questions.

1.4 Authorship

This thesis is based on a project work which was developed by a team of three students. Besides the individually implemented artifacts, the common basis of the project has to be extended to fit the requirements for all further tasks. Therefore, each of the theses will contain a common basis, but every work will answer different research questions including their own prototypical implementations. The main topic *Advanced User Behaviour Analysis* will be divided into the following sub-topics which have their own individual focus:

David Neukam: Case Studies, Automatic Reporting, Evaluation and optimisation of Usability using Advanced User Behaviour Analysis

Maximilian Aster: Repackaging Web Pages using Advanced User Behaviour Analysis

Sebastian Morgenbesser: Web Interaction Archiving and Replay using Advanced User Behaviour Analysis

Between those topics and the overall project several dependencies and relationships arise. Therefore, the chapters 2 and 3 explain the common basis by introducing the terms *Advanced User Behaviour Analysis* and *Mobile Viewport Tracking*. Those chapters were written as collaborative work. Also some minor parts within the introduction chapter such as the methodology were written in collaboration as the same methodology is used by all three theses.

In order to give an overview about the authors of the sections that were written in collaboration, the following table shows all chapters and sections that have different authors. It also includes the own chapters of each author on a top level.

Table 1.1: Author catalog

Chapter/Section	Author
1 Introduction	
1.1 Motivation	Collaborative
1.2 Problem statement	Maximilian Aster
1.3 Aim of the work	Maximilian Aster
1.4 Authorship	Sebastian Morgenbesser
1.5 Methodology	Maximilian Aster
1.6 Related project work	Collaborative
1.7 Relevance for Business Informatics	Maximilian Aster
1.8 Structure of the work	Maximilian Aster
2 Advanced User Behaviour Analysis	
2.1 Introduction	David Neukam
2.2 Aim	David Neukam
2.3 Existing approaches	David Neukam
2.3.1 Eye Tracking	David Neukam
2.3.2 Mouse Tracking	David Neukam
2.4 Heatmaps	David Neukam
2.5 Outcome	David Neukam
2.5.1 Case Studies	David Neukam
2.5.2 Web Interaction Archiving	Sebastian Morgenbesser
2.5.3 Content and process repackaging	Maximilian Aster
3 Mobile Viewport Tracking	
3.1 Introduction	Sebastian Morgenbesser
3.2 Technical Basics	Sebastian Morgenbesser
3.3 Data Model	Sebastian Morgenbesser
3.4 Components	Sebastian Morgenbesser
3.5 Business Logic - Concepts	
3.5.1 Client-side scripting	Maximilian Aster
3.5.2 Action Type Classification	Maximilian Aster
3.5.3 Important Content Identification	Maximilian Aster
3.5.4 Heatmap Drawing	David Neukam
3.5.5 User Tracking	Sebastian Morgenbesser
3.6 Possible Integration into other projects	David Neukam
4 Process and State of the art	Maximilian Aster
5 Repackaging	Maximilian Aster
6 Evaluation	Maximilian Aster
7 Summary	Maximilian Aster

1.5 Methodology

The methodology describes how a certain type of problem is approached. As pointed out by [18] there are in general two distinct paradigms applicable for Information Systems (IS), namely behavioural science and design science. The first one has its origin in natural science and tries to develop and proof a theory, e.g. why a certain human or organizational phenomena occurs when using a specific IS. Typical ways used to justify these theories are case studies, experimentation, or quantitative/qualitative analysis. But this approach is often not applicable, e.g. because the initial situation requires a creative problem-solving approach. In this case design science can support the research process. The difference can be aptly summarised by saying natural science tries to understand ”what is”, whereas design science helps to understand the ”what can be”. The output will be one or more artefacts which are iteratively evaluated to see if they fulfil the requirements defined by the environment. This artefacts can for example be models, methods, or implementations.

> Design science seeks to create innovations that define the ideas, practices, technical capabilities, and products through which the analysis, design, implementation, and use of information systems can be effectively and efficiently accomplished. [18]

An important formulation in this sentence is that the created artefact is called innovation. The intention is to clearly point out that new ways to solve a problem have to be found, and not to use existing knowledge. The usage of existing ”best-practices” would be profession design and would not result in a contribution to the knowledge base of the domain.

Another issue, which is also considered throughout this thesis, is that the results and development of artefacts may be interesting for technical and non-technical persons and should therefore target both audiences. The technical descriptions are therefore detailed enough to understand the implementations and their outcome, in order to allow to use concepts and components for other purposes. For non-technical persons the goal is to clearly define the importance of the problem statement and the utility of the solution. [18]

Key aspects of design science are the three research cycles existing in every design science research project (see figure 1.1).

The relevance cycle shows that the environment opens opportunities and raises unsolved problem statements which have to be dealt with by the developed artefact. But the environment also defines criteria for the acceptance of the artefact. This means as long as they are not fulfilled additional design iterations may be needed. On the contrary, the design result may cause an adaption of the original requirements.

The rigor cycle shows that the incorporation of existing knowledge is essential for the artefact design. This does not only include state-of-the-art theories but also existing artefacts, models, or processes. But the research project should not only use knowledge, it should also contribute to the knowledge base. In this thesis a chapter describing the state-of-the-art in the application context is always preceding the description of the solution.

The last cycle is the design cycle itself. It shows that the creation of an artefact and the evaluation is an iterative process which requires the information gathered through the other cycles,

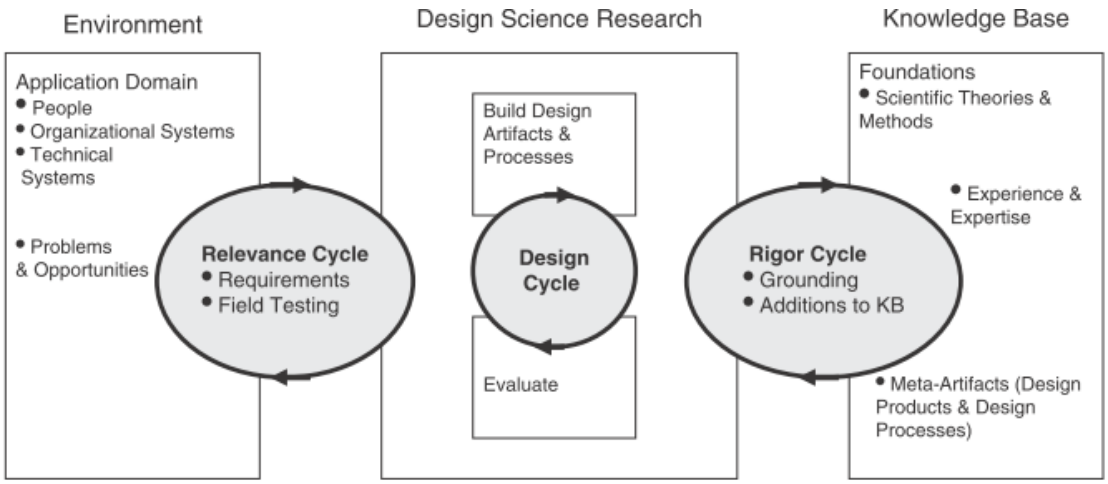

Figure 1.1: Design science research cycles [18]

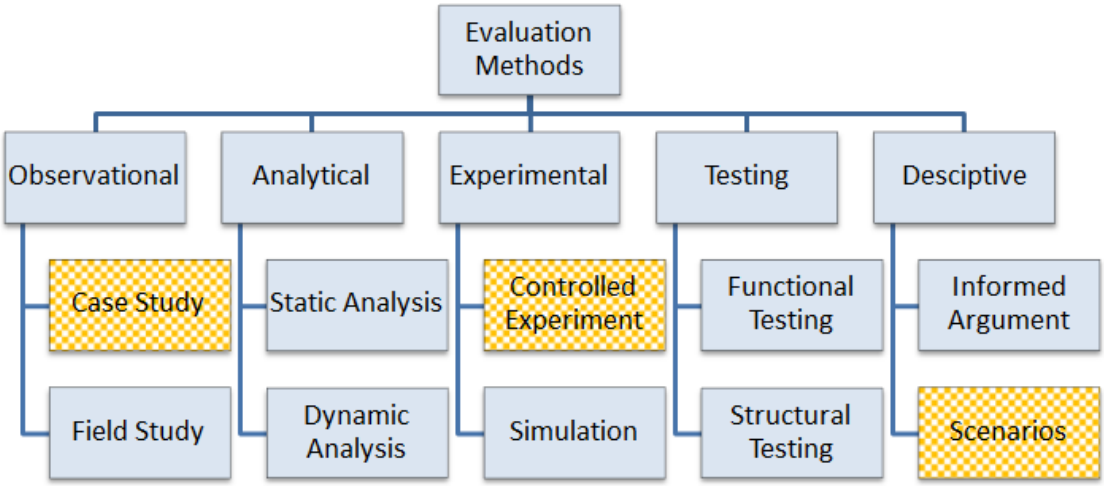

Figure 1.2: Design evaluation methods [18]

i.e. the requirements and the existing theoretical and practical knowledge for the design and the evaluation. The solution part of the thesis and the evaluation are documented in separate chapters.

In [18] various methods for the evaluation of artefacts are presented. Figure 1.2 shows which methods are used to evaluate the results of this thesis. Controlled experiments will be used to show the utility of the solution. In this kind of experiments the artefact is studied in a specific, well-defined environment, e.g. a specific set of web pages within a certain domain. Besides that, scenarios will be used to describe important aspects and additional application possibilities of the artefact. Parts of the solution will also be analysed within a case study to validate the artefact more detailed by a broader audience.

1.6 Related project work

The DBAI group at the Vienna University of Technology initiated several research projects in the area of Web Science up to now. The latest one is TAMCROW [1] which has the goal to develop and propose a model for describing user behaviour of different crowds on the Web. The generation of the model will primarily be based on use cases from web accessibility, mobile browsing, web personalization, and automatic deep web traversal. This model will be applicable to a number of usage scenarios for different user agents. One of these groups are mobile agents. Each of these groups has their own strategies and therefore their own strengths and weaknesses in dealing with Web data. Our project will be used for the mobile browser data acquisition component, and our work contributes as well to the work package content and state/process repackaging. Web Interaction Archiving is a topic mentioned in TAMCROW as a further use case but not a focus of the project. Hence, this new perspective will enrich TAMCROW with additional insights. Finally, our contributions in user behaviour analysis will support the evaluation and help the TAMCROW project to elaborate the differences between the user groups.

1.7 Relevance for Business Informatics

A major focus of Business Informatics lies on the development of solutions for business problems by applying new strategies and technologies. The Web has become an important factor throughout all kinds of business sectors, e.g. for advertising or as a distribution channel. But the target devices are changing from desktop browsers to smartphones, because the market of mobile devices grows, a trend which will hold on for some time. As a result companies have to react and adapt their web solutions in order to be able to compete. Business Informatics combines technical, economic, and social aspects, which makes the Web and mobile devices an ideal platform. The topic of this master thesis tries to integrate these social aspects by analysing user behaviour, and the technical aspect by creating artefacts, designed to solve problems relevant for businesses. Namely, the automatic repackaging and optimization of important content on web pages for mobile devices.

1.8 Structure of the work

Chapter 2 introduces the term *Advanced User Behaviour Analysis* and explains why and how it is different from existing approaches in the field of user behaviour analysis. The following chapter 3 explains the technical concepts and the implementation of the *Mobile Viewport Tracking* project, which was built as concrete approach to *Advanced User Behaviour Analysis* and is capable of tracking and analysing users on the Web.

The chapter 4 describes the state of the art and the theoretical process needed for repackaging. This includes how web pages can be analysed, how they can be modelled, and how the content and an underlying business process can be transformed into a different format.

Afterwards, chapter 5 will explain the implemented artefacts. The first one is a solution for client-side scripting which is able to highlight important content directly on a web page. The

[1] `http://www.dbai.tuwien.ac.at/proj/tamcrow/`

second implementation shows how a repackaging solution transforming a desktop web page into a mobile page can look like.

The evaluation of the process and the artefacts is then done in chapter 6. It will not only show the advantages but will also reflect on limitations. In the end the thesis is summarized in chapter 7. It gives a brief overview about the most important aspects of the thesis and contains an outlook for future work.

CHAPTER 2

Advanced User Behaviour Analysis

2.1 Introduction

The term *Advanced User Behaviour Analysis* is defined as an umbrella term for disciplines that focus on user behaviour analysis using sophisticated technical solutions and measurement techniques. Concrete solutions that fit under this definition are eye tracking and mouse tracking. Web analytics is typically based on static information and dynamic user behaviour [1] of a user on web pages. Many studies using eye tracking technologies were done testing theories like the scanpath theory [20] on websites, identifying determinants and metrics of web page viewing behaviour [34], general user behaviour pattern identification [12] and specific optimisation of web pages [36] targeting the dynamic user behaviour. Static information like server logs is used by [21] to analyse web page viewing behaviour.

As opposed to common web analytics solutions such as Google Analytics[1] which focus only on the path the users followed on a website and lack information about the viewing behaviour on the pages itself, *Advanced User Behaviour Analysis* also analyses the user behaviour on single pages. An example how Google Analytics is used for well-founded web page analysis can be found in [35]. By this definition, also other tracking-methods can be used for *Advanced User Behaviour Analysis* as long as they provide information that goes into the details of viewing behaviour of pages. A new approach that is proposed and implemented in this project is named *Mobile Viewport Tracking* (MVT). The details of the project are explained in chapter 3.

2.2 Aim

The goal of *Advanced User Behaviour Analysis* is to identify actual user behaviour on websites, find out how website visitors interact and identify why they behave in this way. For this, it serves as a basis to identify parts of web pages that are important and those which are not. This could be

[1] `http://www.google.com/analytics`

done on a visual level or on the level of DOM[2] elements. The data collected could be visualised using so called heatmaps which are described in section 2.4.

One target of this project was to create an alternative to eye- or mouse tracking that does not have the drawbacks of both systems. Eye tracking needs expensive and time-consuming studies while mouse tracking suffers from low correlation between eye- and mouse-movement as further explained in section 2.3.2.

This approach could be used to create case studies or projects that build on top of it. Case studies could give hints on user experience on websites, show limitations and give the possibility to improve websites. The recorded data could also be used to build a new kind of web archive that is different from typical crawling approaches such as The Internet Archive [3]. Furthermore, the importance of different regions of a web page could be used to create a repackaged version of this page that shows only the relevant content and omits parts not of particular interest.

2.3 Existing approaches

There are multiple similar existing approaches. The two most common are mentioned below.

2.3.1 Eye Tracking

Eye tracking is an approach to usability studies where the eye-movements and fixations are tracked. The aggregated movements are called scanpath, which is "defined as a sequence of fixations in Areas of Interest or lookzones [34]" which is heavily influenced by the stimulus [20]. These fixation points - defined as "relatively motionless gaze" [34] give a hint what parts of the website were read and which parts were ignored by the user. Website Analysis is typically done with stationary eye tracking systems sold commercially by different vendors [4] [5] .

2.3.2 Mouse Tracking

By tracking the movements and resting points of a participants mouse, a relative estimate of the participants attention can be made. There is a correlation up to 88% between eye and mouse movements [10]. This value is the one usually printed at the homepages of mouse tracking vendors to state the quality of data generated by mouse tracking, but this number is the percentage of "regions that the eye gaze didn't visit" which also "were not visited by the mouse cursor, either" [10]. According to [10], the average correlation is about 58% in sections.

Mouse Tracking is offered by multiple vendors. Some solutions are free[6], others [7] [8] cost from a few Euros a month [9] up to multiple thousand Euro per Website Optimisation project[10].

[2] Document Object Model `http://www.w3.org/DOM`
[3] `http://www.archive.org`
[4] `http://www.tobii.com/`
[5] `http://www.smivision.com/`
[6] `http://www.labsmedia.com/clickheat`
[7] `http://mouseflow.com/`
[8] `http://www.clicktale.com/`
[9] `http://www.crazyegg.com/`
[10] `http://www.m-pathy.com/`

2.4 Heatmaps

Heatmaps are graphical representations of data with an overlay that shows the intensity of data of the specific problem domain like web analytics or meteorology. Every value of the data set is projected on a colour gradient from very high/hot (white or red), to very low/cold (blue).

Heatmaps originate from images created by thermographic cameras or the association of some colours to temperatures. They help identify remarkable data and gain a simple visual impression of a large amount of data in an intuitive way.

In usability study terms, heatmaps are typical representations of recorded data of eye-movements and especially their fixations which show ”distribution of visual attention on the screen” [6] or for clicks in mouse tracking systems.

As a heatmap is a simple visualisation that is very easy to understand, this approach is also used by *Mobile Viewport Tracking*.

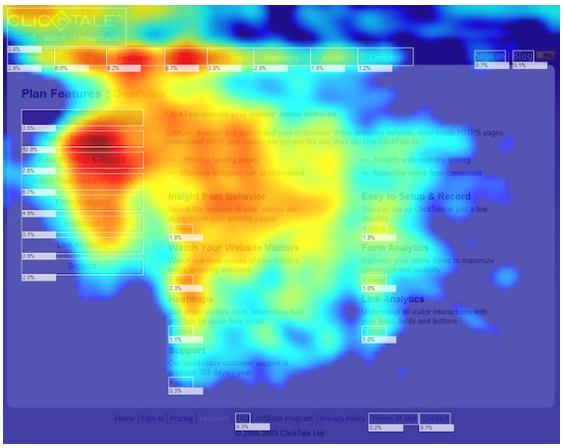

Figure 2.1: Heatmap: www.clicktale.com, Made by Clicktales Mouse Tracking Solution [27]

Figure 2.2 shows a heatmap created with *Mobile Viewport Tracking*, while figure 2.1 is an example of a heatmap created with the commercial mouse tracking software Clicktale [27]. As Clicktale tracks mouse movements that are precise to a single point, these heatmaps show a finer granularity than the heatmaps generated by *Mobile Viewport Tracking* where the whole screen of a mobile device is tracked, but the correlation of attention and mouse-movements is not very high [6].

Figure 2.2: Heatmap: www.gmx.at, Aggregated data from 11 users created with *Mobile Viewport Tracking*

2.5 Outcome

The three different focal points that emerged from the basic idea of *Advanced User Behaviour Analysis* led to three different master theses. These different fields of application are shortly discussed in the following sections. The details can be found in the specific master thesis that covers the corresponding topic.

2.5.1 Case Studies

The first outcome are studies made with the *Advanced User Behaviour Analysis* approach which show possibilities of usage of such a tool. A prototype of an analysis-portal was created that can be used to analyse the recorded data. This tool was used to analyse a study done with students of the course "Applied Web Data Extraction and Integration" at the Vienna University of Technology. Metrics and indicators for web usability are identified that are not available with traditional 'page-level' approaches like Google Analytics. It shows how people behave differently when accessing a desktop or mobile version of a website with their smartphone and what differences and similarities can be found when comparing different websites of one domain to each other. Some Key Performance Indicators (KPI) are theoretically discussed and some were implemented and their results are shown.

2.5.2 Web Interaction Archiving

Another possible outcome of the data from *Advanced User Behaviour Analysis* is a new way to historicise the World Wide Web. A common approach to keep a history of the web is to store entire web pages in an archive at regular intervals. In the future you would be able to take a look at those old web pages and get a feeling how the web actually looked like in the past. The problem is that this approach is missing an important detail of the history which is the user experience. One could say that it is much more interesting to memorize how people actually perceived web pages at different times, rather than just storing the plain content itself. The idea is now to create a mechanism that enables us to archive actual user experience. This is done by using the data that is retrieved from *Mobile Viewport Tracking* which focuses on tracking actions of mobile devices as mentioned before. These "actions" are then combined to "interactions" on specific websites, and so it is possible to reconstruct whole surf sessions of users. There are several ways to visualise those sessions. The simplest form is to generate some kind of textual protocol that describes a specific user session on a specific site at a specific time. A more sophisticated approach, is the rendering of a video that shows the user session as it actually took place in the past in a visual way.

2.5.3 Content and process repackaging

A third outcome is to enhance content and process repackaging. Repackaging means the transformation of a single web page or a multi-page website into another representation. The first kind of transformation would be content repackaging, where parts of a web page are taken and rearranged, for a specific user group or target device, while others are omitted. The second one is

considered process repackaging. In this case the whole structure of multiple web pages and how they are interconnected is transformed, which also affects the underlying business process.

The user behaviour data collected within MVT cannot only support the way the repackaging is done but also leads to new use case scenarios. One of these scenarios is to use repackaging to directly draw a heatmap of the user behaviour within a specific web page. This approach, called client-side repackaging, uses browser functionality to adapt a web page. It is one of the implemented artefacts. The second solution, which was implemented on server side, is more complex and consists of several steps. These phases include the extraction of content from a web page, the segmentation into blocks, the classification and selection of some of these blocks, and the transformation of their content into a mobile version of the website. The clue is to use the user behaviour data to determine important content and to specifically choose this content within the selection phase of the repackaging process. By doing so the generated mobile version for a website makes it easier for users to find key information.

CHAPTER 3

Mobile Viewport Tracking

3.1 Introduction

The last section introduced *Advanced User Behaviour Analysis*, which is the general term for analysing user behaviour especially focusing on web browsing behaviour. This section covers *Mobile Viewport Tracking*, which is the technical instrument we are using for user behaviour analysis. The idea behind *Mobile Viewport Tracking* is quite simple. Because of the pervasion of smartphones these days one could say that a very important part of web surfing behaviour is performed by users utilizing mobile devices. Therefore *Mobile Viewport Tracking* concentrated on tracking smartphone users. The main limitation of smartphones while surfing is still the small screen size, which has a maximum that a user can still handle. This disadvantage for surfing is turned into an advantage for tracking user behaviour. While surfing a user has to make changes to his viewport, which is the visible part of the screen, by zooming, scrolling, panning or rotating his device. The concept of viewport is further described in section 3.2.1. Therefore an assumption can be made, that the different states of the users viewport are in fact content, that the user looked at and with a certain probability also content that he read. A comparison can be made between tracking the viewport of mobile devices and previously introduced eye tracking systems. The aggregation of the recorded data can be an indicator of which parts of websites are more often viewed than others.

So how does the tracking work? The answer is client-side scripting which is explained more detailed in the next section. A JavaScript snippet has to be inserted on the client side, in this case in the mobile device's browser. The script is able to track all the actions mentioned before such as zoom, scroll, pan, rotate etc., and transmits the recorded data periodically to the tracking server. The server receives the data, cleans it, does additional calculations in order to derive the exact action type etc. and then stores it in a database. Later on, calculations and an analysis can be done upon this data base.

How does the script get to the client's browser? There are actually two possibilities. First it can be directly embedded into a target website which should be analysed. The embedding of third party JavaScript components, as in this case, is called beaconing [17]. This is fairly easy,

and it can be done by the websites administrator. The advantage of this approach is that nothing has to be configured on the client side. The website owner has the power to turn on the tracking whenever he wants, of course he most likely would have to inform the user that he is tracked due to existing laws. The second possibility is to use a proxy server to modify the HTTP[1] response of each request returning HTML[2] documents and inject the script. The advantage of this solution is that multiple websites can be tracked, but the user would have to directly accept the tracking and configure the proxy in his mobile devices network settings. In this project the second approach was chosen in order to get a reasonable amount of data from several different websites to analyse and evaluate the project. The next section will cover some technical basics and after that the details of the architecture and abilities of *Mobile Viewport Tracking* are discussed in the following sections of this chapter.

3.2 Technical Basics

This section covers technical basics and shortly explains terms that are used in the following chapters.

3.2.1 Viewport

The viewport is the area that is actually visible during surfing. On a desktop computer the viewport is very large depending on the resolution. Mostly vertical scrolling is needed for surfing. A mobile device cannot display the whole page on the screen, therefore the viewport is always a small part of the website that is currently visible. A lot of scroll and zoom actions are necessary in order to navigate on a classic website designed for desktop computers. The coordinates and the size of the viewport can be retrieved by using simple JavaScript functions that read the following attributes:

- *pageOffsetX*: the X coordinate of the top left corner of the viewport
- *pageOffsetY*: the Y coordinate of the top left corner of the viewport
- *innerWidth*: the width of the viewport
- *innerHeight*: the height of the viewport

[1] Hypertext Transfer Protocol - `http://www.w3.org/Protocols/`
[2] Hyper Text Markup Language - `http://www.w3.org/standards/`

The following image illustrates the viewport on a mobile device:

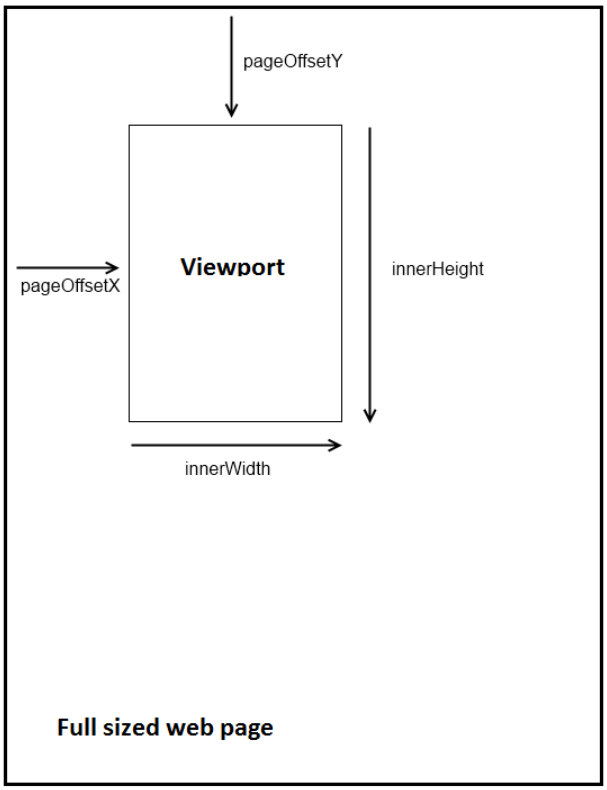

Figure 3.1: Illustration of mobile devices viewport in relation to the full size web page

3.2.2 Web Services

Web Services are used to provide an unified interface to a certain operation over the Web or in a closed network. They are uniquely identified with an URI[3] which provides the access to the service. The communication is very likely performed over the HTTP protocol and the content is transferred in XML[4] format. Basically there are two established types of Web Services. Standard Web Services, often using the SOAP protocol which is described below, and Restful Web Services using standard HTTP functions as an interface.

SOAP

SOAP stands for Simple Object Access Protocol [5] and is an XML based protocol that is used as a wrapper for the communication with the Web Service. As common interface very often a so

[3]Unified Resource Identifier - `http://www.w3.org/Addressing`
[4]Extensible Markup Language - `http://www.w3.org/XML`
[5]`http://www.w3.org/TR/soap`

called WSDL [6] is used. WSDL stands for Web Services Description Language and is also XML based. Every party interacting with the Web Service is able to derive a skeleton from the WSDL in the preferred programming language. The skeleton contains generated classes which make it easy to attach objects as parameters and call methods of the Web Service.

This is an example of the layout of an SOAP envelope:

```
<?xml version="1.0"?>
<s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope">
    <s:Header>
    </s:Header>
    <s:Body>
    </s:Body>
</s:Envelope>
```

The actual message is wrapped in the body tag, whereas the header information is packed into an own header tag. The header is optional and can contain meta information such as encryption or the ultimate recipient.

REST

REST stands for Representational State Transfer and defines a different type of Web Services. Restful Web Services are also available under a specific URI but they respond to standard HTTP GET and POST actions. This makes it very easy to call methods of the Web Service from any device that is able to invoke an URL. The protocol that is used to wrap the Web Services result is not defined and is up to the owner of the service. JSON which is described below, is often used to serialize objects in order to communicate with the service.

3.2.3 JSON

"JavaScript Object Notation (JSON) is a lightweight, text-based, language-independent data interchange format. [...] JSON defines a small set of formatting rules for the portable representation of structured data" [32]. The following block shows a simple example of a JSON object:

```
{
  "name": "MVT",
  "id": "1234567",
  "properties": {
    "number": 13,
    "data": [ "dataA", "dataB", "dataC" ]
  }
}
```

[6] http://www.w3.org/TR/wsdl

3.2.4 DOM

"The Document Object Model is a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents. The document can be further processed and the results of that processing can be incorporated back into the presented page" [44].

3.2.5 Client-side scripting

Client-side scripting is a technique that describes computer programs that run on the client-side, more specifically in the clients browser. The most famous example is JavaScript[7] , which is supported out of the box by the majority of the available browsers. JavaScript and also frameworks based on JavaScript are mainly used to create dynamic HTML pages. The main advantage of JavaScript is, that it can be directly embedded into websites without the need to install client software or plugins to run it. Apart from creating dynamic websites, JavaScript can also be used to communicate with services in the background and also to access DOM properties such as current window size, scrolling position, referring URL etc., which makes it a good choice for this project.

3.2.6 AJAX

AJAX stands for Asynchronous JavaScript and XML and is a technology based on JavaScript, that gives the possibility to reload specific parts of websites without the need of reloading the whole page. This is very useful when creating web applications and dynamic web pages, because the user is not disturbed by page reloads and is able to interact with the web page like it would be a local application. Figure 3.2 shows the difference between the classic web application model and the AJAX technology [19].

[7] `http://www.ecma-international.org/publications/standards/Ecma-262.htm`

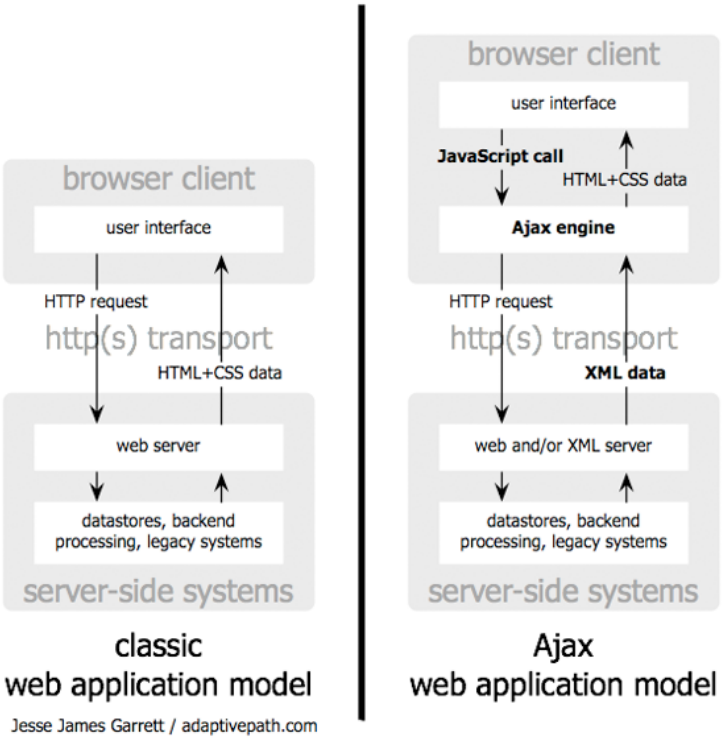

Figure 3.2: The traditional model for web applications (left) compared to the AJAX model (right). [19]

3.2.7 OR Mapping

OR Mapping stands for Object-relational mapping which is used in modern programs to map high level objects and entities to database tables. The layer or program in between is called the OR-Mapper and controls the communication from and to the database. In practice a programmer can e.g. call a save method of the OR-Mapper, pass an object as parameter, the OR-Mapper than translates the objects data to the databases query language and executes it. The result is then again parsed, turned into an object and returned to the caller. In the Java world the most famous OR-Mapper is called *hibernate*[8].

3.3 Data Model

This section shows the basic data model used for storing and analyzing the recorded viewport data:

[8] `http://www.hibernate.org`

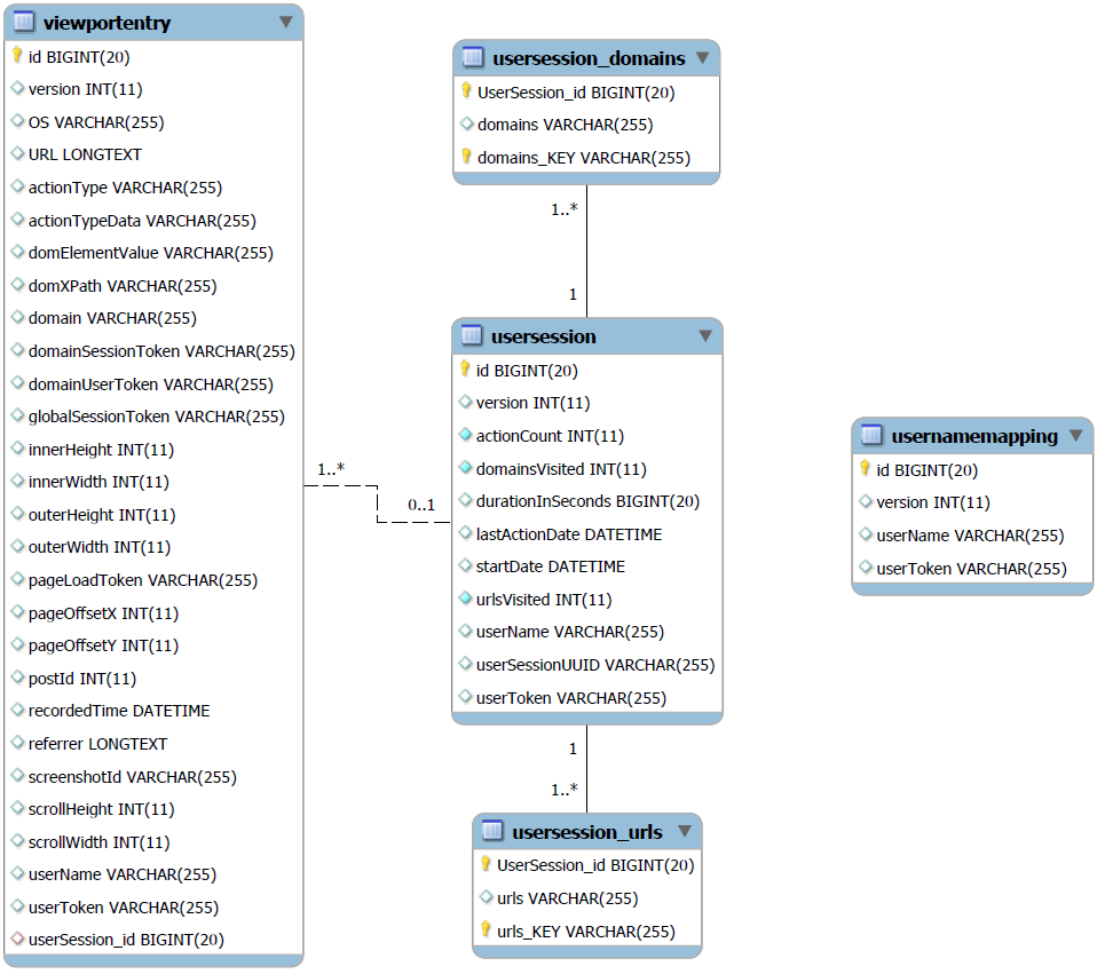

Figure 3.3: Data Model - part 1

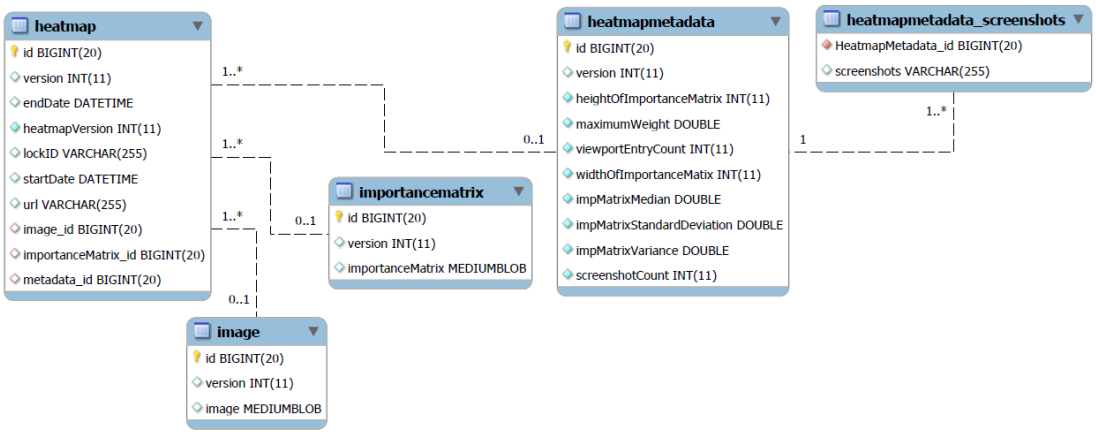

Figure 3.4: Data Model - part 2

The three main entities are the following:

- *ViewportEntry*: This is the main entity. A ViewportEntry represents one action on a mobile device for example a scroll, a zoom, a pan etc. The data is a flat table allowing the usage of data mining techniques.

  Important fields are:

  - *actionType*: Type of the action such as scroll, a zoom, a pan etc.
  - *domElementValue/domXPath*: If the action was key press or click the relevant DOM element is recorded together with the path and the actual value of the element
  - *domainSessionToken/domainUserToken/pageLoadToken /globalSessionToken*: Various session tokens in order to track the surfing user
  - *pageOffsetX/pageOffsetY*: The coordinates of the north west corner of the viewport. This represents the scrolling location on the website
  - *innerWidth/innerHeight*: The width and height of the viewport. This indicates the zoom level
  - *referrer*: Referring URL e.g. google.com
  - *recordedTime*: The exact date and time when the action occurred
  - *screenshotId*: If a screenshot was made for this action, the field contains an unique id which relates to the taken screenhot image
  - *userToken/userName*: If the user can be tracked (accepting cookies), the fields contain the unique session id and an optional user name if the user has registered himself

- *UserSession*: This entity is used to wrap up many viewport entries related to the same user session into one package. Additionally meta-data is added to the user session such as visited domains/URLs, the start time of the session, the end time of the session, the action count of the session or the duration of the session.

- *Heatmap*: This entity is used to store a heatmap based on the aggregated data of multiple sessions performed on one specific website. The heatmap contains an importance matrix that represents the raw data of the calculated heatmap itself. Together with a screenshot, a graphic representation of the heatmap can be generated from the raw data.

3.4 Components

This section will provide an overview of the architecture of the architecture of the *Mobile Viewport Tracking* system, and will shortly describe each component and its purpose within the system.

3.4.1 Architecture

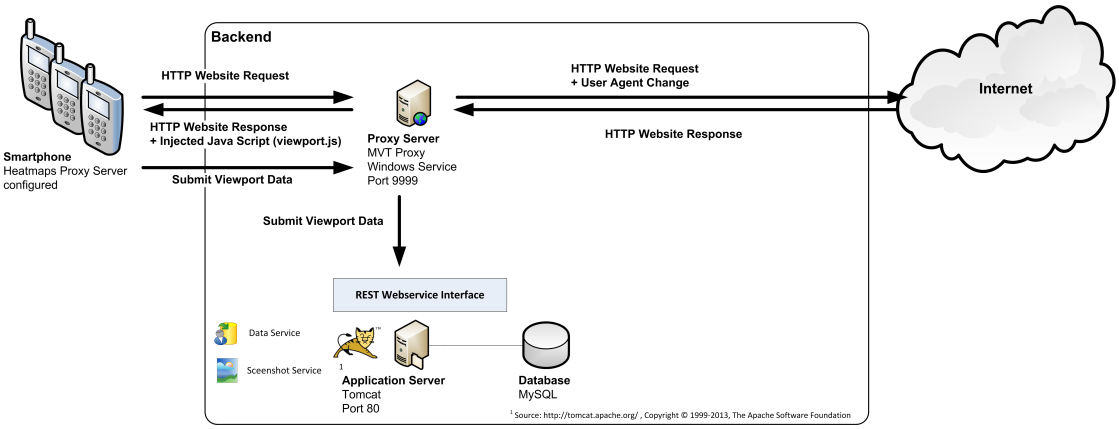

Figure 3.5: Architecture Overview

3.4.2 Client-side script

As can be seen in the architecture diagram the client side is is most likely a smartphone running a mobile browser. To be able to track various actions related to the surfing behaviour of the user such as pan, zoom, scroll etc. client side code has to be injected somehow. The client-side logic basically consists of a JavaScript file containing functions for tracking surf related actions and sending the recorded data to the central application server which stores it in the database. The script can either be embedded directly in the target web page or has to be injected by a proxy server (see section 3.4.3).

3.4.3 Proxy Server

The proxy server is used for modifications of HTTP requests and also HTTP responses which are triggered by an user who is surfing on his mobile device. Therefore the proxy has to be configured on each mobile device that is going to be tracked. For HTTP requests the proxy server modifies the user-agent of all incoming requests, in order to avoid getting mobile websites which are not

targeted within this project. The current used user-agent is an iPad user-agent which proved to be the best suited one for this studies as technologies as Flash are also not available on an iPad but the screen size is mostly like a desktop device.

On the HTTP response the proxy modifies the HTML content and inserts a JavaScript tag that is importing the client-side tracking script into the head section of the HTML page. The proxy server is based on the Brazil Framework and is a fork of PAW Proxy [9]. The configuration of the proxy on the supported mobile platforms can be found in the appendix section A.2.1.

3.4.4 Web Service

The Web Service component is the general interface of the application server. It serves as a data collector and provides a store data action as Restful Web Service method. The underlying business logic (see section 3.5) is responsible for cleaning, preparation, analysis and classification of the received data. The whole server side is written in Java[10] and published as web application on a Tomcat[11] server. The responsibilities of the service basically split into the following two parts:

Data Service

As already mentioned, this service is responsible for storing and also processing the incoming data. The method is called directly from the clients (smartphones) as HTTP POST request. It also provides methods to retrieve heatmap data or to trigger the creation of heatmaps.

Screenshot Service

The second part of the service is responsible for taking screenshots of the visited web pages. This is done with Selenium Web Driver[12]. This tool provides the possibility to open up a browser with a specified URL, waiting for the website to load and finally taking a screenshot of the entire web page which is later used to create heatmaps. For each user that surfs to an URL an screenshot has to be taken. Therefore the service is managed by a simple queuing mechanism with first-in first-out principle in order to prevent overloading of the server if to many people use the service.

3.4.5 Database Layer

The last component is the lowest layer which is responsible for storing the processed data and providing it for later analysis. For database connection and operations the OR-Mapper hibernate[13] is used. Beneath this a MySQL database is attached, but because of the usage of an OR-Mapper, the database is easily exchangeable.

[9] http://paw-project.sourceforge.net/
[10] http://www.java.com/
[11] http://tomcat.apache.org/
[12] http://docs.seleniumhq.org/
[13] http://www.hibernate.org/

3.5 Business Logic - Concepts

3.5.1 Client-side scripting

The essential component for the collection of user data on client-side is implemented in JavaScript. It can either be embedded in a web page by the content provider itself or by using a proxy server to inject it. The second approach has the advantage that data about any web page can be collected, the disadvantage is that the configuration of a proxy server is necessary, which may not be possible on all mobile devices.

One problem which occurs if the script is injected into every page that is loaded, is that it is also executed for IFrames within a page, which do usually just contain advertisement. In those cases the execution is aborted and no data is collected within this frame, because the data is not considered relevant for this project.

The script does not require that JavaScript frameworks like jQuery are included. The only dependency is to Hammer.js[14], a JavaScript library to track multi-touch gestures, which is directly included in the main script. All the functionality of *Mobile Viewport Tracking* is defined in an own namespace *MVT*. This allows a clear structure and avoids conflicts with other client-side code may running in a web page.

The first action taking place after the initialization is to identify the user, for this purpose cookies on different levels are used. The lowest scope of user-tracking is per page, followed by domain, and global tracking. For a detailed explanation see section 3.5.5.

Afterwards the actual action tracking is started. The script always keeps track of the last web page state it sent to the server. This state basically includes the current position of the user on the web page, i.e. the current viewport. The next step is that this state is periodically compared with the current state, if a change occurred the data is queued to be sent to the server. Apart from this periodic data, data is explicitly collected e.g. for clicks. Some properties which are always collected are the URL, the operating system of the device, X and Y position, or the inner and outer width and height.

Table 3.5.1 shows a list of basic events that were identified and by which DOM events they are triggered. These events represent the explicit actions which are tracked and recorded, e.g. if a user submits a form (SUBMIT) or if text was entered into an input field (INPUT_CHANGED). Some of the events can also be specific for the device which is used (e.g. WEBSITE_DEACTIVATED). The table also shows one usage of the Hammer.js library, namely the tracking of DOUBLE_TAP and HOLD actions. Compared to the periodically collected data these events are delivering more information, e.g. the XPath of the HTML element which was invoked, the element value, or the position where the event was triggered.

When tracking user input security and privacy issues arise. For example when tracking the user behaviour on an online banking web page. To avoid possible fraud, password inputs are not recorded. But unfortunately it is not possible for a user to verify what is tracked and what is ignored. It has to be mentioned that the injection of the script is not possible if the connection is secured, i.e. if HTTPS using SSL or TLS is used.

[14] `http://eightmedia.github.io/hammer.js`

Table 3.1: Mapping of DOM events to MVT events

MVT event	DOM event
CLICK	click (a, area, span, div, p)
SUBMIT	submit (form)
INPUT_CHANGED	change (input, textarea, select)
KEY_PRESS	keydown (input, textarea)
WEBSITE_ACTIVATED	pageshow (window) focus (window)
WEBSITE_DEACTIVATED	pagehide (window) - iOS blur (window) - Android
ROTATION_TO_HORIZONTAL	orientationchange (window)
ROTATION_TO_VERTICAL	orientationchange (window)
DOUBLE_TAP	doubletap (hammer)
HOLD	hold (hammer)

As mentioned before, the collected data is not directly sent to the server but gets queued to reduce the server and client load. For asynchronously sending the encoded data a *XMLHttpRequest* object is used. It uses a HTTP POST request to transmit the data to the web server. An important aspect is that the same origin policy does not allow the access to the data returned from the MVT web server, because that server is not the host server of the web page. Nevertheless the data is received on the server. One way to circumvent this policy is the usage of JSONP[15] which is a technique embedding a new *script* element into the HTML code with a specific request URL that usually returns a function call with the response data as parameter. For the purpose of tracking user actions it was not yet necessary to return data, but it could be a future extension. An alternative would be the proxy server which can redirect requests and manipulate the headers to circumvent the same origin policy.

Filters

Passwords were filtered on the client-side (see appendix section A.1.3 line 222ff). Only the first letter is transmitted in clear text, other letters are replaced by asterisks, in order to make aware of the capability to record these passwords. In fact code to record passwords could be injected by every node on the route between the webserver and the client.

Major explicit content websites were blacklisted and a filter for specific keywords was included on the server side to anonymize data recorded on this sites.

3.5.2 Action Type Classification

The next step after the client-side collection of data (section 3.5.1) is the classification of the collected data. A basic classification was already done within the client-side script. This classification, which was shown in table 3.5.1, will now be extended on server side. Every

[15] JSON with Padding - `http://www.json-p.org`

Table 3.2: MVT events identified on server side

MVT event
ZOOM_IN
ZOOM_OUT
SCROLL_UP
SCROLL_DOWN
SCROLL_LEFT
SCROLL_RIGHT
SCROLL_UP_LEFT
SCROLL_UP_RIGHT
SCROLL_DOWN_LEFT
SCROLL_DOWN_RIGHT

viewport change which was tracked has a unique, consecutive ID. This allows to easily find the previous viewport data and to compare the new one with it. To determine if a ZOOM_IN or ZOOM_OUT has happened the *innerWidth* and *innerHeight* are compared. To identify scrolling in one of the eight possible directions the X and Y position of the old and new viewport are compared. The source of the Action Type Determiner can be found in the appendix in section A.1.2.

3.5.3 Important Content Identification

After the data is collected (see section 3.5.1) and post-processed (see section 3.5.2) it is necessary to bring it into a format for further usage, this includes a machine-readable representation and a human-readable one. Unfortunately the raw data is not directly usable for further reasoning or evaluations because most actions on the user interface of a web page will create an event and the granularity of these actions is too low. The data is therefore summarized with the goal to identify the most important content on a web page. The approach chosen within the MVT project is described in the following section. The condensed data is then made available by a Web Service method returning a JSON object which can also be deserialized to a Java object. A human-readable version of the data are heatmaps, which can be automatically drawn (see section 3.5.4).

Weighted raster

Important content is identified by assigning weights to specific parts of the web page. For this purpose the web page is rastered, i.e. a virtual raster with the same height and width as the web page is used to represent the actual page. Every cell in the raster has a predefined width and height which can be configured. All cells are initialized with 0 weight. This raster will be further called *importance matrix*. For every user action which was recorded the weight of that part of the raster will be increased. Depending on the type of MVT event and other aspects a different weight will be assigned.

Weighting

The *importance matrix* is created by iterating all entries for a certain web page. The algorithm is first explained considering a simple scroll event to an arbitrary location on a web page. First of all, the cells which are affected are determined, i.e. the start column and row, and the number of spanned rows and columns. Two basic factors for the resulting weight of the event are readability of the text within the viewport, and the time spent at this specific location to study the content. The pressure representing the zoom level is calculated by looking at the longest dimension of the viewport entry, i.e. the inner width or height depending on the rotation. After controlled experiments with different devices, values for *minimum readability* and *maximum readability* were defined. If the longest dimension lies within the interval of these two values the pressure gets a high weight, otherwise not. Additionally the weight increases the closer the dimension value is to the *maximum readability*. To calculate the time spent on the specific part of the web page, the difference of the current and the previous viewport change is evaluated. Similar to the readability an interval with the highest weight was evaluated through controlled experiments. This interval ranges from 1 to 20 seconds and corresponds with the findings of [20] and [34], which stated that important information is processed during the first 15 seconds on desktop computers.

Afterwards the weight is low again, because the assumption is that the user may have put the smartphone away or closed the browser.

After calculating these basic weights, more specific calculations are performed depending on the action type of the previous and the current event. In general the weighting is divided into point-weighting and area-weighting.

Listing 3.1: Area-weighting

```
+--------------------+
|10|10|10|10|10|10|10|
+--------------------+
|10|10|10|10|10|10|10|
+--------------------+
|10|10|10|10|10|10|10|
+--------------------+
|10|10|10|10|10|10|10|
+--------------------+
| 5| 5| 5| 5| 5| 5| 5|
+--------------------+
| 5| 5| 5| 5| 5| 5| 5|
+--------------------+
```

Area weighting means that all cells within the viewport get the same weight added to their already existing weight. But there is an exception to this rule, namely in case the current rotation is vertical, in this case the weight of the last $1/3$ of the cells from the bottom of the viewport is halved. The reasoning for this exception is that a user usually starts to scroll after the first part of the page is read and does not continue reading until the end before scrolling. This was done because experiments showed that the last third is not read but people started scrolling after $2/3$ of the content was read. An schematic example is shown in listing 3.1. This method is applied for a simple scroll event.

Listing 3.2: Point-weighting

```
+--------------------+
|  |  |  |  |  |  |  |
+--------------------+
|  | 3| 5| 5| 5| 3|  |
+--------------------+
|  | 5|10|10|10| 5|  |
+--------------------+
|  | 5|10|15|10| 5|  |
+--------------------+
|  | 5|10|10|10| 5|  |
+--------------------+
|  | 3| 5| 5| 5| 3|  |
+--------------------+
|  |  |  |  |  |  |  |
+--------------------+
```

Point-weighting on the other hand always has one cell in the center which gets the full weight. The weight of the surrounding cells is steadily decreased. The outer corners for example are always $1/5$ of the original weight. The radius of the point-weighting could be adapted, but a value of 2 cells showed good results. An schematic example is shown in listing 3.2.

The following list shows what kind of weighting is performed for which event, or which combination of events:

- The current event is a CLICK or HOLD
  In both cases a point-weighting is performed. The weight to be used is the pressure multiplied with a certain value for HOLD respectively CLICK.

- The current event is a ZOOM_IN
  This event causes an area-weighting of the cells in the viewport. The weight is higher than when using DOUBLE_TAP to zoom because the zoom factor is multiplied with the basic weight of this action. The zoom factor is in this case not the absolute zoom factor of the web page but the relative one, compared to the previous viewport entry.

- The previous event was a DOUBLE_TAP followed by a ZOOM_IN
  A ZOOM_IN caused by a DOUBLE_TAP is a common action resulting in point-weighting but with less weight than a ZOOM_IN using a multi-touch gesture. This is done because a DOUBLE_TAP is sometimes performed on a specific area of the web page which could be of interest but often also just to increase the zoom in general.

- The previous event was a ROTATION_TO_HORIZONTAL
  In this case a area-weighting is performed because a rotation to horizontal orientation indicates that the content was interesting enough to give it more space on the display. A rotation to vertical on the other hand does not result in a weighting because the user most likely changed the orientation after finishing reading. The reason to check the event type of the previous event is, that the rotation events do still contain the previous width and height which makes it necessary to wait for the next event.

- The current event is a scroll event
  If the event is one of the eight scroll events an area-weighting is performed with the pressure and time factor as basis.

Implementation

The Java class representing the heatmap in the MVT project is called *Heatmap* and contains information about the web page, i.e. the URL, information about the validity of the heatmap represented by a start and end date, a screenshot of the web page, metadata, and the *importance matrix*. If a heatmap is created, some of these properties are initialized directly, e.g. the URL, and some asynchronously, e.g. the image is set as after the screenshot service (see section 3.4.4) generated it. The creation respectively the update of the heatmap is performed as soon as new data is retrieved from the client-side script and the post-processing was performed. The default case is that the same *Heatmap* object is used for all users, but it is also possible to generate specific heatmaps, e.g. just with the data of a specific user. These objects can also be retrieved by invoking the Restful web service.

Besides that, heatmaps are versioned, i.e. the current version of a heatmap can be archived and if new data is retrieved a new heatmap is created. At the moment there is no automatic mechanism to archive heatmaps, but it could be a future extension. If a web page changes too much, the identified important content loses its validity.

3.5.4 Heatmap Drawing

For drawing heatmaps, two different algorithms where created that both have their benefits and drawbacks. While the Raster Drawer is closer to the implementation and more technical, the Colourful Drawer has it's focus on a pleasant graphical representation. The sources of the concrete implementation of both drawers are available in the Appendix A.1.1.

Figure 3.6: Comparing the Colourful Drawer (left side) with the Raster Drawer (right side)

Raster Drawer

As it was mentioned in section 3.5.3, *Mobile Viewport Tracking* uses a simple segmentation approach where the page is split into very small tiles (20px x 20px in this example). The Raster Drawer draws horizontal and vertical lines where the tiles are split. It also prints the logarithmized weight of this tile into the cell. The value is logarithmized with base 10, so the simple calculation of

$$weight = 10^{cellValue} \tag{3.1}$$

is needed to get the original value.

For visual representation, this drawer uses the opacity to represent the importance of a cell. It takes the ratio of the current cells value and the maximum weight of the heatmap to determine the opacity. A high opacity means a high importance, a low opacity means a low importance. This relationship is linear, so it is a simple percentage-value from the maximum.

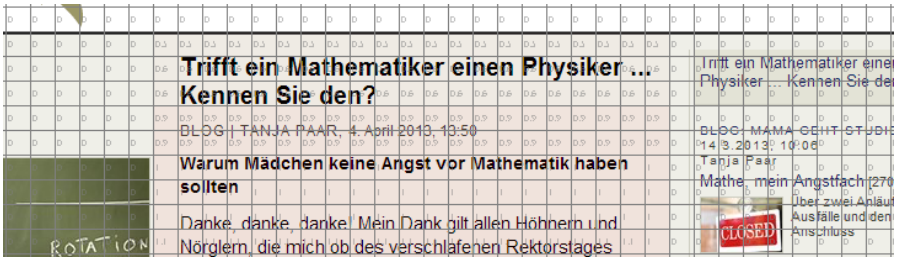

Figure 3.7: Raster Drawer in detail: the logarithmized weight and the opacity

As there is no projection of the values onto a colour gradient, it is not a heatmap according to the strict definition in section 2.4. The Raster Drawer can be used to see the real values of the heatmap.

Colourful Drawer

The Colourful Drawer has it's focus on a pleasant graphical representation. It maps the value of the cells to a colour spectrum seen in figure 3.8.

Figure 3.8: The colour spectrum used to map the value to the resulting colour

The data is smoothed graphically by using a circle brush to create colour-gradients seen in figure 3.9.

Figure 3.9: The circle brush used to smooth data

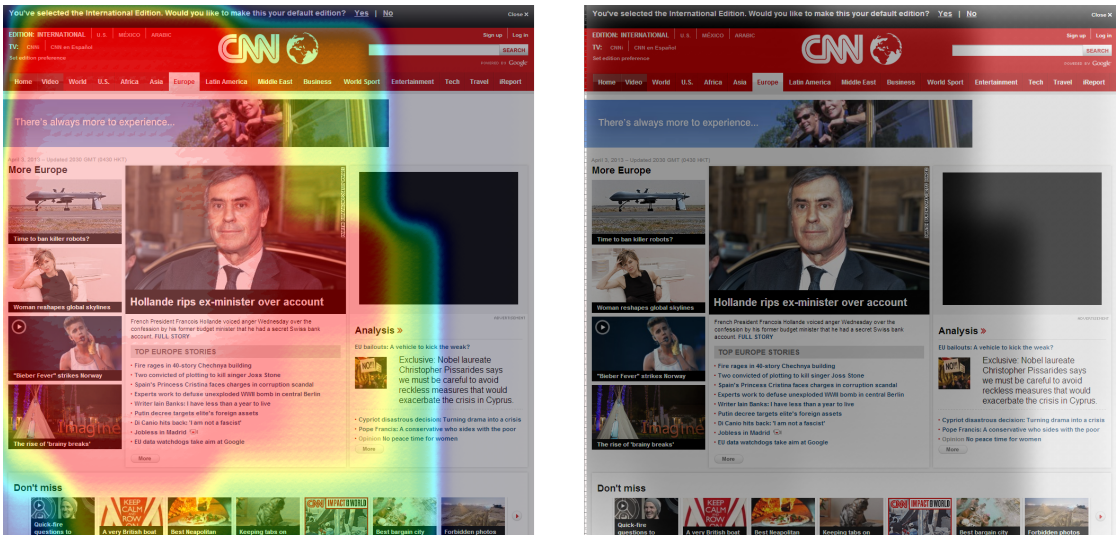

Figure 3.10: The colour scale is mapped from a transparent black and white image with different alpha-values (right side) to the colours of the colour-spectrum (left side)

First, an empty monochrome image is created (see figure 3.10). At the center of each tile, the centered brush is added with the relative weight (current weight / max weight) set as the opacity. In this manner, a smoothed version of the heatmap explained in section 3.5.4 is created. This

monochrome smoothed heatmap is mapped linearly onto the colour-spectrum shown in figure 3.8. The whole transformation is shown in figure 3.10. This leads to the final heatmap.

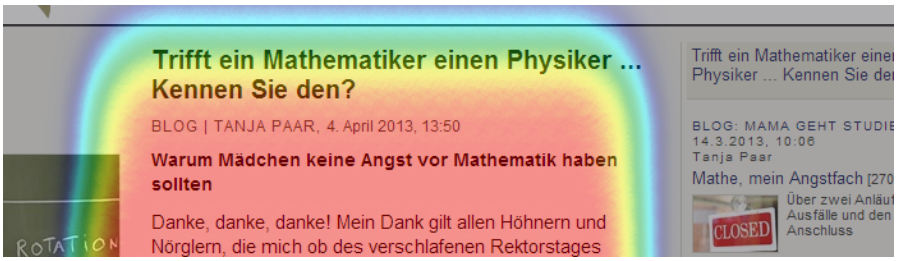

Figure 3.11: Colourful Drawer in detail

3.5.5 User Tracking

This section focuses on the business logic that handles the tracking of users. If the many viewport changes that occur could not be matched to a specific session even if it is only per page or domain, no useful analysis can be made. Therefore a tracking mechanism has to be used in order to group specific actions together and pin them to a specific session. The tracking of users is established using cookies. If users would not allow cookies than tracking would just be possible on the page level. The following section basically describes the different scopes of user tracking and their purpose within the tracking strategy.

Page

The first session identifier that has to be introduced is not a cookie. It is called page load token and it represents an unique id that is regenerated every time the page is loaded or reloaded. This means that also an F5 key press would trigger the creation of a new page load token. The simplest form of grouping actions together is by a single page load, which is exactly what the page load token is used for. This makes it possible to analyse all actions that took place on a specific page e.g. scroll down, zoom in, scroll, zoom out and so on. After the URL is changed by the user, or reloaded a new page load session begins.

Domain

The domain session token is actually a cookie. It is created for a specific domain to identify a user surfing multiple pages within a domain, including sub-domains. This ensures that different surfing sessions of the same user on the same domain can be tracked. Listing 3.3 shows the creation of a cookie which is available across sub-domains. In this example, the value of the cookie could be read on all pages within the ”tuwien.ac.at” domain, e.g. `http://www.tuwien.ac.at` or `http://www.dbai.tuwien.ac.at`. All actions performed on these pages would belong to the same domain session. If the user changes the domain by e.g. clicking an external link, another cookie is created for the new domain if it does not already exist. Besides that, the session has a limited span of life, the default is 30 minutes. A special case which always starts a new

domain session, is if the referrer, i.e. the page the user is coming from, belongs to a different domain. The reason to create a new session is that the user may come to the page by following a direct link, e.g. from a search engine. In this case the actions performed by the user are seen as single sessions.

Listing 3.3: Domain cookie creation

```
1 document.cookie = "MVT=12345; expires=...; path=/; domain=.tuwien.ac.at"
```

Another token used on the domain level is the domain user token, which is similar to the domain session token, except that it has no timeout. This means it is an eternal cookie that is only deleted if the user decides to delete it manually. The cookie is used to identify an user that has once surfed on a specific domain and then later comes back to it and visits it again. It offers the possibility to track a specific user forever but restricted to a specific domain.

Global

The main problem of all the approaches that were mentioned above, is the inability to track a specific user over multiple websites which would enable interesting evaluations and analysis. The reason why this does not work or should not work, is because of the limitation of cookies. They can only be created on domain level and not for different domains. Fortunately, the usage of the proxy server, which is configured on the mobile devices of all tracked users, opens other possibilities to track users in another way.

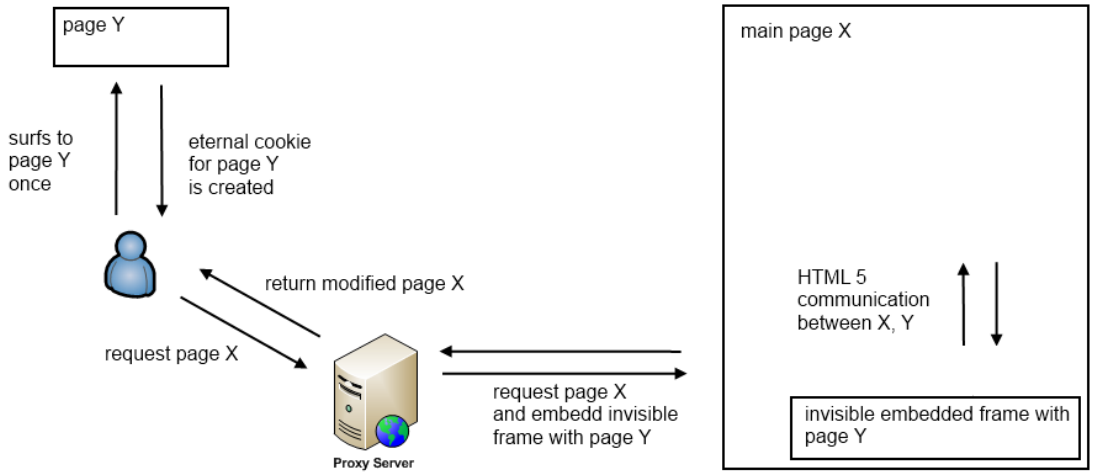

Figure 3.12: IFrame embedding trick using HTML5

Figure 3.12 shows a trick which enables the system to track users globally or across multiple websites. It uses the ability of the proxy server to change the content of requested websites. Mainly what the proxy does is, it requests the content of the desired web page and embeds an

invisible IFrame into the page that refers to our main registration page, which is called page Y in the figure. The user just has to visit the registration page Y once in order to create an eternal cookie for it, and after that he can be tracked. Every time the user visits a web page, the proxy server injects the invisible IFrame of page Y. In the client-side script it is now possible to communicate between the main page and the invisible IFrame using the HTML5 function *postMessage* and therefore retrieve the eternal cookie and identify the user uniquely.

This unique identification is a prerequisite for the server-side generation of the global session. As long as a user is performing an action on any website the same global session is used. Only if he becomes inactive and continues surfing at a later point in time a new session is created.

Registration Page

The registration page was already mentioned in the last section. It is used for creating an eternal cookie and also to give the user the possibility to register himself using a nickname. The registration page enables the user to map his eternal cookie to a self chosen name or number. This can be useful for later analysis of specific users or user groups. For Android users the global tracking cookie can be created automatically, without the users need to visit the registration page with the default system security settings. iPhone users have to visit it once, because otherwise the cookie is rejected by the devices operating system. Figure 3.13 shows a screenshot taken on an iPhone while surfing to the registration page.

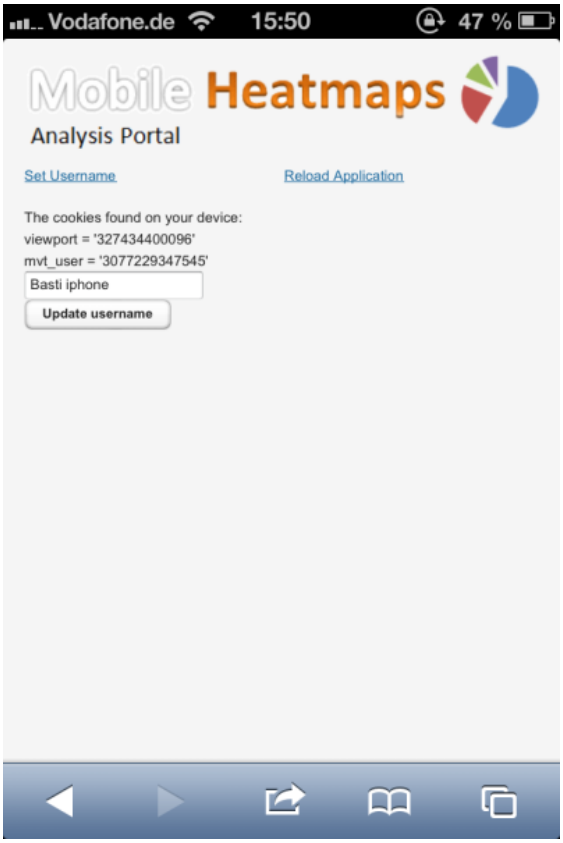

Figure 3.13: Registration page

3.6 Possible integration into other projects

Mobile Viewport Tracking was created under the premise of high integrability.

The core implementation is created as a Java library from which the drawing functions, business object design and persistence can be easily reused.

By using this library, a simple SOAP or REST based service interface could be created to expose data to other programming languages without the need of recreation of an underlying data access layer. There are innumerable usage scenarios for such a service.

Several REST methods are already provided by the current implementation of *Mobile Viewport Tracking*, e.g. a service method giving back the importance matrix of a given URL to be used for further website processing and repackaging.

As RDF[16] and OWL[17] are flexible data representation languages, a transformation from the given database to a triple store should be simple. With such a transformation, all benefits from

[16]Resource Description Framework - `http://www.w3.org/RDF`
[17]Web Ontology Language - `http://www.w3.org/OWL`

using such a triple store could be used as exposing the data to the open data initiative or querying the data with powerful languages like SPARQL[18] .

The analysis web application offers the possibility to export all data to a simple CSV[19] file. This CSV file is in flat data format which could be used as an input for common data mining tools. To ease usage, the transmitted data is already stored with enriched information (action type, session IDs) that could be used when the target is to improve such classifications (e.g. by a machine learning algorithm to determine the type of action).

[18] SPARQL Protocol And RDF Query Language - `http://www.w3.org/TR/sparql11-overview`
[19] Comma Separated Values - `http://www.ietf.org/rfc/rfc4180.txt`

CHAPTER 4

Process and state of the art

For the last couple of years trends in software development led from applications produced for desktop machines to web applications and mobile devices. This evolution will most probably continue within the foreseeable future. But we already reached a state where users do not rely on a single source of information anymore and may simply leave a web page if they do not find the desired information. That is just one of the reasons why it is important to stick out of the mass. To support this task *Advanced User Behaviour Analysis* can be used. It allows to validate quality criteria of a web page or to make a detailed evaluation using heatmaps, for example by viewing the surf behaviour of representative web page users. This way it is possible to improve and optimize the page. But what if these optimizations would not have to be done manually. More precisely, the collected data is used as input for an automatic approach to adapt or optimize a web page for specific target devices. This kind of transformation is called repackaging. As a concrete scenario, one can transform a desktop web page into a mobile-optimized representation. This repackaging targeting mobile devices could be very valuable for people and companies who do not have the time and money to develop dedicated mobile versions of web pages but do still want to support them. And as surveys show there will be immanent need to focus on these devices [37].

To have a systematic approach to this problem a process model is introduced. The various phases which were identified are displayed in figure 4.1. Although not all sub-phases have to be considered for a solution, at least the main phases Analysis, Modelling, and Transformation have to be considered. The first task is the extraction of the web page content, followed by the analysis of the page to classify the content, and the selection of the relevant content parts. Afterwards this parts have to be stored in an appropriate data model, which may not only be capable of showing relations of parts within a web page, but also between pages within the same domain. If this is achieved the web page can not only be transformed, but the whole process or structure underlying the web page can be transformed.

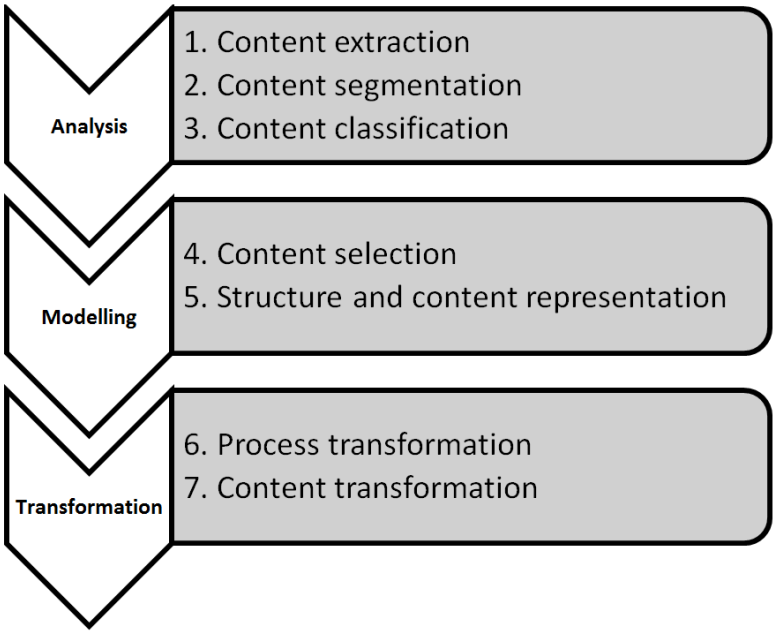

Figure 4.1: Repackaging process

The repackaging of a web page can basically be done in three ways:

- *On-the-fly*
  As soon as a user requests a web page in a certain target format the whole repackaging process is executed for that page. An optimization for this kind of generation could be a caching mechanism which keeps the repackaged web page for available for a certain time span
- *Periodic generation*
  The generation of the repackaged version is done periodically within a configured interval. The generated version is then saved and indexed to be able to find and provide it immediately when requested
- *Change triggered*
  The repackaged version is generated as soon as the source web page changes. In this case the repackaged web page is not up to date anymore. Same as for periodic generation, the generated version is then stored to allow immediate access

This chapter will investigate the state of the art within the Web Science fields required to accomplish and implement the proposed repackaging process. Before starting to describe the actual process a general classification of web pages is done. Afterwards the main phases of the proposed process are handled within own sections. At first the Analysis phase, where besides of content extraction, segmentation, and classification also the role of Web Mining is discussed. The following section will then describe the parts of the Modelling phase, i.e. the selection of content

and how it can be represented using existing tools and frameworks. The next section will then explain the last phase, the Transformation phase. This includes not only the transformation of web page content but of a whole underlying business process.

Afterwards an overview of existing repackaging tools is given, and their functionality and used concepts are compared. The chapter then concludes with an outlook of future challenges.

4.1 Classification of the Web

The Web has undergone a tremendous evolution towards a large-scaled graph where content is highly interconnected. This was supported by big technologies, especially the releases of HTML and CSS standards[1]. All web pages have different characteristics, but to have a common understanding and vocabulary in the following parts the criteria proposed by [42], namely complexity, interactivity, and functionality will be used for a classification. *Class 1* web pages are usually simple, with few functionality and no user interactivity. These pages can also be called static pages. *Class 2* pages have a higher complexity and functionality with limited or no user-interaction. These pages may use client-side scripting but do not use it for the modification of the page, but just to add dynamic features like effects. They can be seen as *class 1* web pages enhanced by client-side scripting. *Class 3* contains web pages with the highest complexity and functionality. Besides that, they may contain a lot of user-interactivity. They usually employ a high number of technologies to dynamically create their content, e.g. server-side technologies like JSP, PHP, or servlets. And they may use client-side scripting to adapt their content. Pages belonging to *class 3* will synonymously be called web applications or client-built pages. A web application using AJAX would allows that the whole application consists of one single page with a single URL. The basic operating principle of AJAX was described in section 3.2.6.

What is nowadays known as Web 2.0 is a part of the Web that is dominated by human communication, e.g. Wikis, Blogs and social networking sites. Using Web 2.0 applications, the reader does not only consume information contained on web pages, but also creates and shares content with others to become a creator. The content created in this context is called user-generated content. Relating Web 2.0 to the classification done before, it can be said that it corresponds to the criteria interactivity and therefore spans *class 2* and *class 3* web pages.

4.2 Analysis phase

4.2.1 Web Mining

The goal of Web Mining is to identify and extract relevant data from web pages. According to [16] it can be divided in three main fields, namely Web Content Mining, Web Structure Mining, and Web Usage Mining. All of them can be used within the Analysis phase of the repackaging process.

Web Content Mining is interwoven with web page segmentation 4.2.3 and classification 4.2.4 because its idea is to identify useful data and knowledge from a web page. It uses machine

[1] `http://www.evolutionoftheweb.com`

learning, data mining, and linguistic techniques to analyse content. A typical use case is to determine the relevance of content for search engines in order to improve the quality of their results.

Web Structure Mining incorporates the fact that the Web is a graph. A basic idea is that a predecessor page, i.e. the page containing the hyperlink which was used to access the current page, provides additional information to classify the current page. This may result in a richer classification because the descriptions may come from different authors, or because of the fact that the page is referenced by a lot of other pages.

Web Usage Mining analyses information of user interactions with a web page. This information could come from click-stream analysis, i.e. tracked user clicks, or *Advanced User Behaviour Analysis* (see chapter 2).

4.2.2 Content extraction

The first phase of the repackaging process is the extraction of content which serves as the basis for the web page segmentation. Depending on the approach to be used for the segmentation, the extraction has to deliver a different input format. The XY-Cut, which is further described in section 4.2.3, for example often uses a document image as input for the further segmentation. Most other techniques are using HTML as input. At that point it is important to make the differentiation between parsed and rendered HTML. An HTML parser creates an internal representation of the HTML code, like the Document Object Model (DOM), and it is possible to extract data using APIs e.g. by using DOM traversal and accessing node properties. But visual information is only available if a web page is rendered, which is computationally more expensive than just parsing the HTML because all the style information has to be taken into consideration and also client-side scripting may result in a visual change, often called reflow.

At the moment there are a lot of HTML parser for Java available. Some of them are based on the W3C DOM API[2] and deliver a standardized org.w3c.dom.Document after parsing the HTML, e.g. HtmlCleaner[3] or JTidy[4]. But there are also tools available which go their own way, like jsoup[5] which allows advanced extraction methods similar to jQuery selectors.

HTML renderer for Java are very scarce. Two promising and active projects, also claiming to support newer standards like CSS3, would be CSSBox[6] and HtmlUnit[7].

A major problem is also that web developer do not verify that their web pages are working with those renderer, but optimize their content just for the major browsers used by their customers. That is a reason why also the integration of these browsers, which are implemented for end users, is an appealing approach. One popular tool in this area is Selenium[8] which allows to control web browsers by using the browser's built in support for automation. It already provides the required WebDriver API implementations for all major browsers. The browser is then controlled remotely

[2] http://www.w3.org/TR/DOM-Level-2-Core/
[3] http://htmlcleaner.sourceforge.net
[4] http://jtidy.sourceforge.net
[5] http://jsoup.org
[6] http://cssbox.sourceforge.net
[7] http://htmlunit.sourceforge.net
[8] http://www.seleniumhq.org

and it is possible to execute any action, including user actions like for example clicks, and to retrieve data and HTML elements. This actions can either be implemented using the designated WebDriver API methods like *findElement* or by executing arbitrary JavaScript methods.

Other issues are that content changes over time, or may not be accessible. The first problem means that web pages are not static but their content can change very frequently, e.g. news pages. This means caching of extracted content and furthermore the transformed version of a web page may not always be possible, but an on-the-fly extraction will result in a significant performance decrease. In order to be able to cache a web page a reliable method to detect changes is needed. This is a problem by itself, because the DOM structure of web pages is usually different on every page visit, e.g. because the advertisement which is displayed keeps changing. One possible solution for this issue is to calculate the similarity of the DOM trees, as for example done by [30], to decide if the content change is significant enough to extract the content again.

The second problem was the access to content. A lot of web pages require a user to log in to see content. If the tool trying to extract the content would just open the URL of the page where content has to be extracted, the information would be of no use. If the web page owner itself is hosting the repackaging solution trying to access the content, it would not be an issue because it would be in his own interest. Therefore a closer look will be thrown on the first case. Web servers usually use session cookies to identify a user and to show the content he is allowed to see. Hijacking the users session by stealing this session cookie, or the tracking and recording of the users credentials on the other hand would represent a security and privacy issue. As this problems seems not to be solvable, it might be interesting to look how other services, especially search engines which are living of data, are handling this issue. It shows that their crawlers are also not able to index protected pages because they too lack the possibility to log in. What Google did, was to establish a cooperation with the owner of the protected content. The owner makes the content available for them, and on the contrary Google allows its search engine users to access the first protected page which was found for free (First Click Free program [9]). That way Google can offer better results and the content provider gets more visibility. Similar incentives could be found if content provider would allow content extraction for a repackaging solution. The problem of accessing content is also discussed in the section dedicated to Process repackaging 4.4.4.

4.2.3 Content segmentation

The main purpose of the content segmentation resp. web page segmentation (WPS) is to identify content parts on a web page which logically belong together. According to [45] three different approaches can be used to tackle this problem. The first technique uses rules to determine the different segments of a web page. But it was pointed out that these rules are mostly domain-specific and rules designed for general purpose do not lead to applicable results. The second group of solutions uses machine learning techniques. The drawback of this approach is that a high amount of data is required by the learning algorithm in order to perform well. The most promising category is the last one, heuristic-guided methods. This intuitively makes sense, when considering Gestalt theory, which is described in section 4.2.3, as a set of heuristics applied by human beings to identify and group visual information. Most approaches are working with heuristics, although

[9] `http://support.google.com/webmasters`

not necessarily with Gestalt theory, to find the content of a web page which belongs together. The following paragraphs give an overview about the content within this section.

A first step to web page segmentation is the determination of the reading order of elements on a web page. In [31] an approach to determine this reading order by using the XY-Cut algorithm 4.2.3 is discussed. In their paper the document type is not restricted to web pages but any electronic documents. Besides that it discusses the issue that the original source format of a lot of documents is not available anymore although it is needed for new kinds of target devices. A key interest was further the conversion of the document content into an own XML format which reflects the correct logical resp. visual structure of the content. Both issues can be ignored when using an HTML web page as source format, because the visual information can be extracted from the rendered web page, and the source format, in this case HTML, is also available. But the underlying problem of determining the page reading order is still essential and has to be considered when performing web page segmentation.

A method of information extraction from HTML documents based on visual information using a combination of web page segmentation and information extraction was proposed in [5]. They pointed out that an approach just considering HTML code without visual information is not robust because of the possibilities that come along when combining it with CSS. Their solution creates an abstract representation of a web page. Therefore the document is rendered and treated as a set of boxes, afterwards the page segmentation algorithm is used to find visual areas and their hierarchy within the document. In the end the information extraction is performed.

One important statement, which can be found in several approaches, is that the HTML and the DOM tree are not appropriate enough to perform web page segmentation or web page understanding on them because a lot of visual information is just available after rendering. Besides that, the DOM tree of modern RIAs is often not reflecting the real structure of a web page. For example *DIV* elements are typically used as containers but their content may be displayed dynamically or at a completely different position on the page than the DOM structure would suggest. The more CSS is used the less important the structure becomes. Because of the same reasons the presence of a specific tag is also not enough to make assumptions about the resulting segmentation, although some tags are more likely to cause a segmentation, e.g. headings. But which kind of visual information should be taken into consideration. Unfortunately, reasoning based on CSS classes is not a reliable method to determine the intention of certain elements because it is not standardized in any way. On the other hand there are approaches available which work without the overhead of rendering the visual information by just focusing on text properties, i.e. the text density of block elements on a web page. Depending on this calculated value block elements can be grouped and the page segmented. The algorithm is explained in more detail in section 4.2.3. [23] [22]

Existing theoretical work about web page segmentation and web page understanding often relies on Gestalt theory. In [45] especially the four laws proximity, similarity, closure and simplicity were considered for an approach to repackage web pages for mobile devices. The algorithm described in their paper calculates and compares values for the considered Gestalt laws to segment a page. An important aspect of this approach is that the information content is not reduced and that the original page layout is kept. The problem of keeping the whole information is that it may be too overwhelming for a user. Unfortunately, the selection of specific content

requires manual annotation or the incorporation of user behaviour data. [45]

Other approaches like the Web Page Processing System (WPPS) [38] are based on an ontological model consisting of an underlying physical, visual, and logical model. The processing starts with a web page analysis resulting in the physical model, followed by web page understanding techniques and information extraction aiming to create a logical model which can be further transformed for the usage in external applications.

A client-side web page modification tool to supplement existing Wikipedia articles with additional content was described in [33]. Besides that, especially in the field of web accessibility, e.g. the Web Accessibility Initiative (WAI)[10] driven by the W3C, client-side solutions like browser plugins or bookmarklets are used to clean and optimize web pages for disabled people.

Existing tools repackaging web pages are either based on manual repackaging approaches, i.e. content has to be configured and updated by the creator of the mobile web page, as for example done by Jaemobi[11], or are focusing on specific sources like RSS feeds or WordPress to re-package that content, e.g. Mippin[12]. An semi-automatic approach is for example used by DudaMobile[13], which analyses the page structure and creates a mobile-optimized web page out of it.

[23] points out some additional applications of web page segmentation in the field of Information Retrieval. Namely for text classification, de-duplication and full-text search. Especially their non-visual approach allows faster computation, and therefore easier integration into those application areas, than other solutions.

Gestalt theory

Gestalt theory is a concept trying to explain how human beings perceive visual information and how visual elements are grouped together based on certain aspects. There are six major laws in Gestalt theory such as similarity, proximity, closure, symmetry, continuity and common fate. [43] [40]

The first basic law is similarity. Elements are assumed to belong together if they have similar characteristics, in the context of web pages, elements having a similar layout are perceived to belong together. Another more concrete example would be the design of radio buttons or other form fields where every element has the same basic design.

[10] http://www.w3.org/WAI
[11] http://www.jaemobi.com
[12] http://www.mippin.com
[13] http://www.dudamobile.com

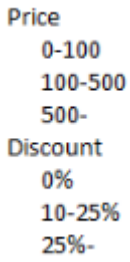

Figure 4.2: Law of continuation

The second principle is continuation. It says that a reader is lead to another element using certain visual cues like curves, or pointers towards something. For web pages the law of continuity is responsible that elements are perceived as belonging together if they are within the same row or column. Besides that it supports the creation of hierarchies if the continuation stops, e.g. the headers in a list of elements are breaking the flow (as shown in figure 4.2).

Figure 4.3: Law of closure

The next principle is closure, it is about providing enough information for humans to fill the remaining parts using their imagination (as shown in figure 4.3. A common example is logo design where parts of the text or image are omitted but people are still able to see the whole picture by filling the gaps on their own.

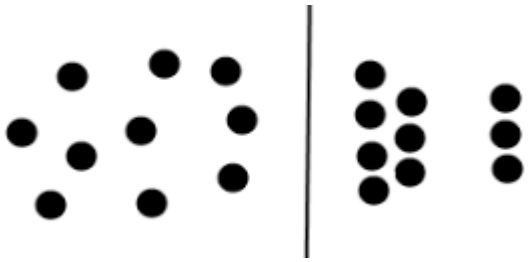

Figure 4.4: Law of proximity

Another law is called proximity which says that elements are perceived as a group if they are placed close together. This can be seen in figure 4.4. The left part shows elements which are not

in proximity with each other and therefore not perceived as a group. In the right part the elements are forming two groups where they are close to each other and have a high proximity.

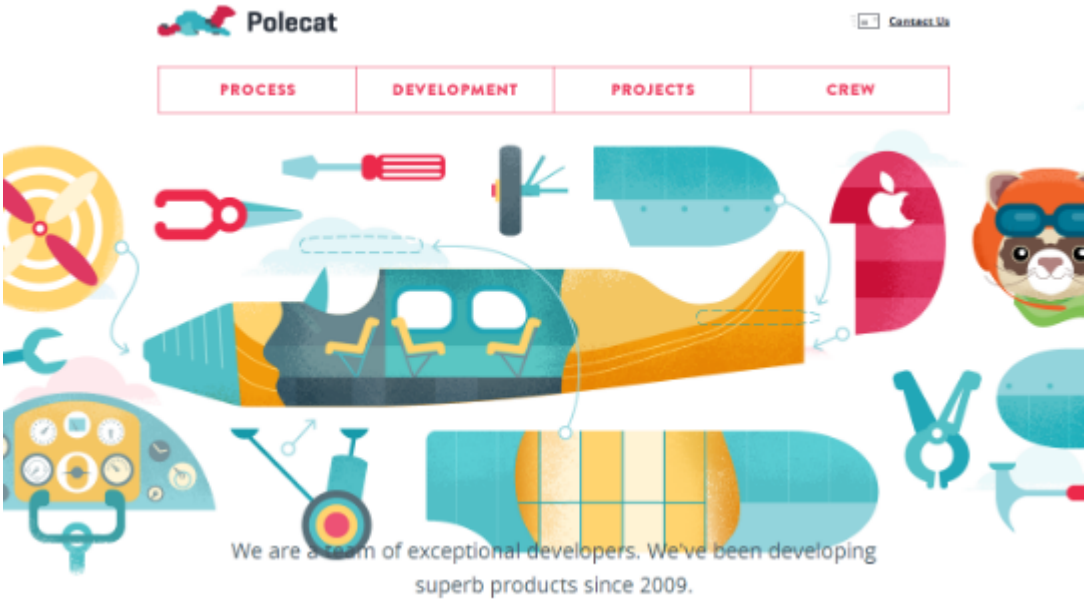

Figure 4.5: Asymmetry in web design[14]

The law of symmetry is a very general principle which states that humans are more comfortable with symmetric objects than with unbalanced. Basically, something is seen as symmetric if there exist two halves of the same size and form around a central point. Symmetric elements are easier to remember and we can get a faster overview in comparison to asymmetric elements where the user first tries to find out what is wrong. This does not mean that web design is always symmetric because asymmetry can be a tool to generate interest (see for example figure 4.5).

The last law, the law of common fate states that objects moving with the same speed and in the same directions as others are perceived to belong together. Web pages can easily integrate dynamic features using JavaScript or HTML5 but also static images can imply movement using visual cues. Examples for dynamic action would be drop-down menus or tooltips. All the items within the menu and also the content of tooltips are perceived as content belonging together if they appear in a dynamic way. [43]

XY-Cut

The idea of the XY-Cut algorithm is mainly based on concepts of Gestalt theory. The input format for the algorithm is usually a document image and not a structured format. Nevertheless especially the applicability for web page segmentation will be assessed. The basic step which is recursively performed is to search for empty areas between blocks of content, e.g. text or

[14] `http://www.ipolecat.com`

images, entirely crossing the viewed segment. For these blocks the algorithm is applied again as long as the empty area is bigger than a certain threshold. The reading order of the text blocks is country-specific, but for western countries the reading order usually follows a top-bottom left-to-right path. [31]

Figure 4.6: XY-Cut ordering

Figure 4.6 shows how the XY-Cut segmentation is performed. In the beginning the only gap spanning the web page is the one separating the new parts (1) and (2). The search for an empty space is then continued for both elements. This results in two new sub-blocks for both of them.

As pointed out in [31] there exist three main problems, namely the ”L-Shapes”, the threshold parameters and the cutting strategy. Because of the underlying box model of HTML and CSS all elements are rectangular but it is still possible to create shapes where the algorithm can not be applied in a straightforward way (see figure 4.7). The second problem is the definition of useful minimum and maximum values for the empty areas used to split blocks. The third problem is the cutting strategy which depends on the type of document. It was pointed out that an approach which always cuts the block with the biggest empty area in between delivers good results for web pages. But for other documents, e.g. technical documents with multi-column layout this can lead to errors because a horizontal cut may be performed before a vertical one separating the columns. Out of this reasoning they adapted the XY-Cut algorithm to use a score function preferring vertical cuts that span multiple consecutive blocks.

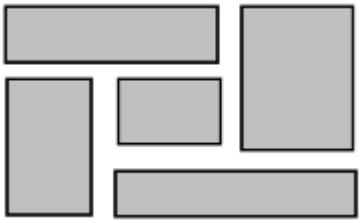

Figure 4.7: Example of an L-shape

Vision-based Page Segmentation (VIPS)

The idea of the Vision-based Page Segmentation algorithm also relies on the visual representation of content blocks. The basis for this algorithm is the DOM tree of a rendered web page. First, it searches for basic objects which can not be divided within the DOM tree, e.g. text nodes or images. The result of the algorithm is the so called vision-based content structure where these basic objects are grouped according to rules based on visual perception. These groups do not necessarily have to correspond with a node in the DOM tree. For every group the Degree of Coherence (DoC) is calculated depending on how consistent the parts of the block are. The value ranges between 0 and 1, whereas 1 means the elements have the same characteristics. The granularity of the blocks can be influenced by changing the threshold of the minimum DoC, the Permitted Degree of Coherence (pDoC). If this value is increased it will result in smaller blocks, because all the children of the blocks have to be more similar to each other. [8]

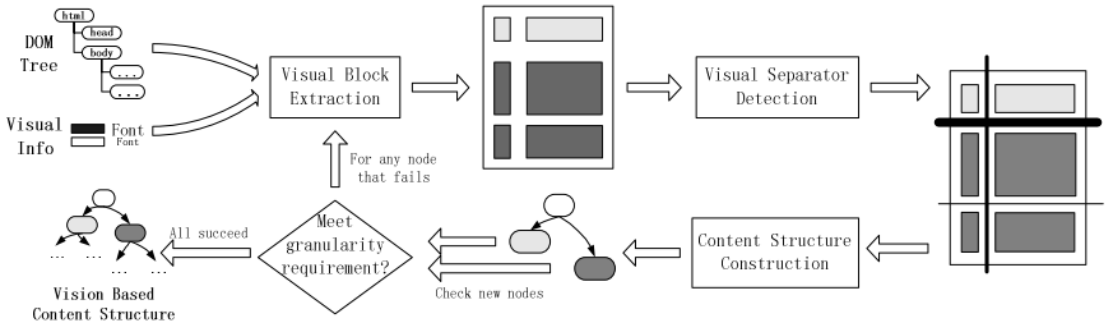

Figure 4.8: VIPS algorithm [7]

The algorithm (illustrated in figure 4.8) starts at the root node of the DOM tree and checks for every node if can be further divided, if that is not the case it is put into the pool of basic objects. The next step is the detection of visual separators, every possible separator gets a weight depending on the differences with its neighbours, e.g. colour, distance, font style. After that a hierarchical structure is created. Starting with the lowest weight two blocks of basic objects are added as children of a new visual block covering both of them. This is done for all separators. Afterwards the most granular visual blocks, i.e. new leafs of the hierarchical structure, are

checked if their DoC is higher than the pDoC. If this is not the case, those blocks are further processed. If all leafs are meeting the requirements the algorithm outputs the final structure. [7]

Another approach [39] describes a hybrid solution based on a combination of the VIPS and XY-Cut algorithm. By doing so, this DOM and image based technique tries to improve the quality and relevance of the segmentation.

Web Page Processing System (WPPS)

The Web Page Processing System (WPPS) developed by the Vienna University of Technology [38] is a Java-based framework which allows to perform web page segmentation on a higher abstraction level than other approaches. The underlying model of the framework is called the unified ontological model (UOM) and consists of nine layers. They are visualised in figure 4.9.

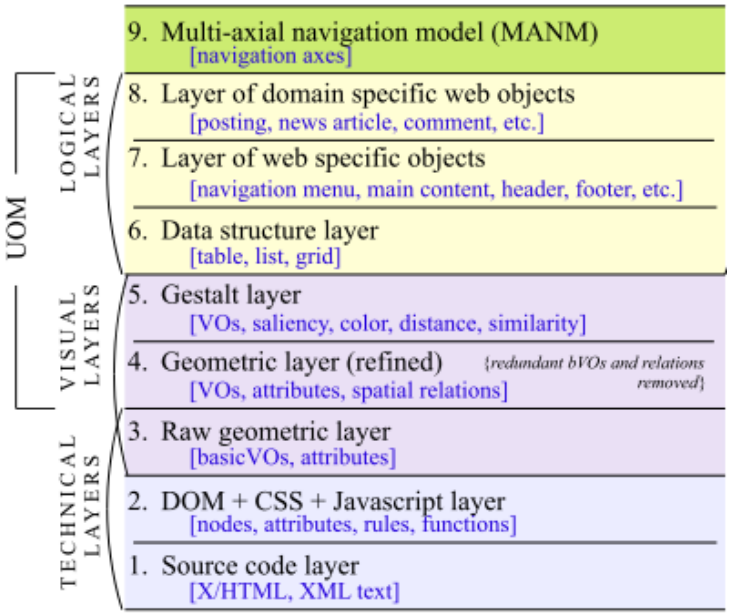

Figure 4.9: Layers of the unified ontological model [24]

The first layer is the HTML source code followed by the second layer which also considers the DOM tree, CSS, and JavaScript code to create a set of basic visual objects. The next layer calculates raw geometric values like the position, size and content. The fourth layer adds relationship properties like distances and alignment. Layer five uses Gestalt theory to identify figures which are perceived to belong together. The idea is, that the segments are the same as if humans would split a web page in significantly different parts. To determine if blocks are similar or if the principle of continuation can be applied, style properties like the colour, and distance values are evaluated. The sixth layer uses generic knowledge about documents to identify objects such as lists or tables. The following layer further applies web specific knowledge to find, for example, navigation menus. Layer eight makes use of domain specific knowledge to find

specific segments like an article on a news page. The last layer is designated to use the gathered information to repackage a web page for other target groups. The layers 3-5 can be described as visual layers whereas the layers 6-8 represent logical objects. The frameworks provides the possible to create own wrapper resp. enricher which map and group visual objects to logical objects. [38]

To improve repackaging capabilities the framework goes beyond DOM structures to determine the ordering and nesting of segments by introducing the multi-axial navigation model (MANM). It allows to define axes consisting of certain logical objects and a direction. An example for such an axis would be a navigation menu axis where the individual menu entries are lying on the same horizontal line.

Block Fusion algorithm

In contrast to the previously described techniques, the Block Fusion algorithm described in [23] is a linguistic approach based on text properties. The main purpose of this algorithm is to find the main content of a web page. A big advantage of this approach over visual-based techniques is that it is not necessary to render the page, because it is just based on text properties. This allows a much faster computation and makes it possible to be used in scenarios where time is critical, e.g. within web crawling. The basic assumption is that the transition between segments is caused by changes in the density of text blocks, e.g. from blocks with low density containing only a few and shorter words like in navigation menus, to blocks with higher density like product descriptions on a shopping page or the main article on a news page. They define text density as the number of words within a specific 2-dimensional area. Therefore a fixed width is assumed for the block and the height are the number of lines which would be shown if the text would be inserted into the block using word wrap. The ratio between the number of tokens and the number of lines of a text block b is the text density $p(b)$.

$$p(b) = \frac{Number\,of\,token\,in\,b}{Number\,of\,lines\,in\,b}$$

To further improve the accuracy, the last line of blocks with multiple text lines should be ignored.

What the Block Fusion algorithm does, is that this density is calculated for all text blocks. In the next step always two blocks are compared and as long as the second density lies within a certain range around the first density the blocks are merged. Merging means that the two blocks will become a single one and its text density gets calculated. This is repeated until no further merging is possible. A special case which significantly increases the performance of the algorithm is to also merge blocks where the density of the predecessor and the successor is identical and the density of the block under consideration is lower. This pattern can often be seen on web pages, e.g. if there is a headline in between two parts of a news article. In this case all three blocks will be merged to one. Another finding which is incorporated to improve the performance of the algorithm is that main content is usually surrounded by template text. If that is not the case it is a strong indicator for template content with higher density, e.g. a copyright text.

Besides the text density the link density can be used to find the main content. Link density is defined as the ratio of the number of token in a text block and the number of tokens belonging

to the link. This means a block containing mostly links has a high link density. One could for example reason that the main content will have a lower link density than template parts.

An open-source library of the algorithm was implemented by the author of the paper and is called boilerpipe[15]. The basic use case of the tool is to remove the template parts of a web page to identify the main content. This technique can be used to improve the quality of search engines by only indexing the relevant content of a web page without parts belonging to the template. The following example 4.1 shows how boilerpipe can be used. The output of this listing is a list of blocks with their corresponding text density. In the result it would be possible to see that the density of template blocks is < 10, whereas the density of main content is > 10.

Listing 4.1: Boilerpipe example

```
1 URL url = new URL("http://en.wikipedia.org/wiki/Web_page");
2 TextDocument doc = new BoilerpipeSAXInput(HTMLFetcher.fetch(url).toInputSource()).
      getTextDocument();
3 String text = DefaultExtractor.INSTANCE.getText(doc);
4
5 for (TextBlock block : doc.getTextBlocks()) {
6         log.info(block.getTextDensity() + "\n" + block.getText());
7 }
8 log.info("Main content:\n" + text);
```

4.2.4 Content classification

After a web page has been segmented the corresponding parts have to be classified. This task is hard to solve in a way which covers all possible web pages. Therefore heuristics are often chosen. These heuristics can either use visual information, e.g. a block with different contrast in the upper part of a page could be an indicator for the navigation menu, HTML attributes or structure, e.g. elements with a *class* or *id* attribute value containing ”nav” may also indicate the navigation bar, or text properties, e.g. blocks having a high text density may indicate that a segment contains the main content. All those heuristics can also be combined to improve their quality.

The most reliable classification is the one that was already given by the developer of the web page. In that context the WAI-ARIA (Web Accessibility Initiative - Accessible Rich Internet Applications) should be mentioned. It describes how semantics can be added to HTML elements. The proposed way is to use the *role* attribute for that. If a HTML element defines this attribute and it contains the value ”navigation”, the usage of heuristics is not necessary anymore. A list of roles can be found in the WAI specification[16].

4.3 Modelling phase

4.3.1 Content selection

The selection of content depends on the further usage and target format of the repackaging. For example, if the target is an optimized version for screen readers, specific parts like images can be

[15] https://code.google.com/p/boilerpipe

[16] http://www.w3.org/WAI/PF/aria/roles

skipped.

This phase will often require manual effort in order to create a good transformation result. A tool might for example show a preview of the resulting web page, and offer the possibility to select or omit content for the final transformation. Furthermore the content which is selected could be added by the framework used for the transformation, e.g. like manually adding images or the possibility to share the content on a social network. An example is shown later when having a closer look at existing tools.

4.3.2 Web Modelling

Complexity does not only increase in traditional desktop software development but also in the field of web engineering. Therefore proper software design becomes important for web applications too. Within the field of Web Modelling several visual notations to describe web pages appeared. The only standardized modelling language among them, which has been adopted as a standard by the OMG, is the Interaction Flow Modeling Language (IFML)[17]. This standardization is an important step because one of the critics of Web Modelling is that customers do not want to get into a locked-in situation by a set of models and technologies which are only known by a few people. The IFML is very much based on experience with the Web Modeling Language (WebML)[18], which it is also going to successively replace. The main objective of IFML is to express content that is visualised on the user interface and to define the connection to the business and persistence layers of an application. It further allows to model the navigation paths between sub-pages and the user interactions within those pages. A huge advantage is that it defines different perspectives of the UI in a formal way without going to the implementation level. This allows to generate the front-end separately for specific devices. A simple example using the basic concepts is shown in figure 4.10. The user initially sees a list of product categories (*CategoryList*), as soon as one of them is selected, the products within this category are displayed (*ProductList*). If the user also selects an entry in that list, a redirect to the detailed page of the product is taking place (*ProductDetails*). [4]

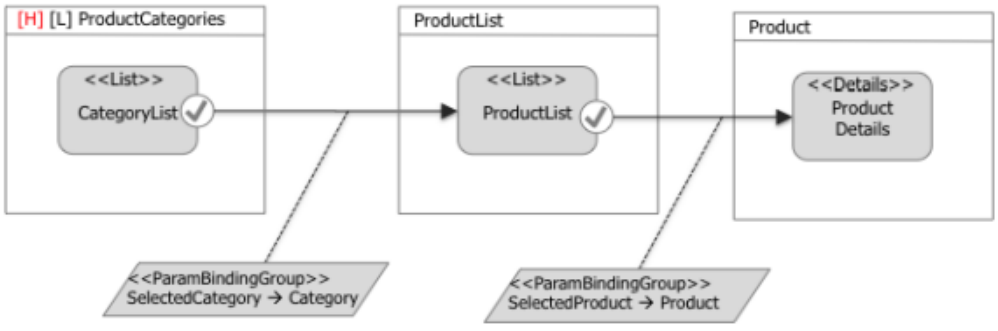

Figure 4.10: Exploration of products example[19]

[17] http://www.ifml.org
[18] http://www.webml.org
[19] http://www.ifml.org/wp-content/uploads/IFML-Bookstore-Example.pdf

The first question should be where the information, events, and entities to be modelled come from. A field of web development also concerned with this problem is GUI testing. Currently it is a task requiring a lot of human effort in order to create appropriate test cases. A typical approach is to record the actions performed within the application and replay these actions within the test cases. A test case is successful if the actual result is the same as the recorded one. The most common approaches to identify elements to be invoked, and to validate the result of the test case are either based on image comparison, e.g. Sikuli[20] or XPath validation, e.g. Selenium. The creation of these images used for the comparison and robust XPath expression, for example checking for the ID of elements, is manual effort. If it would be possible to create a model of the GUI, and a test designer verifies the validity of this model, the creation of test cases could be automated based on this model. The task of creating this model of an existing web page is sometimes called GUI Ripping. Other use cases where this model could be useful apart from GUI testing were pointed out by [29]. Namely, porting legacy application to new platforms or the development of model verification tools. [29]

Another research field which can provide useful concepts and implementations for the field of Web Modelling is Web Crawling, especially the crawling of web applications using AJAX. The current problem of web crawlers is that content which is loaded using AJAX can not be indexed because crawlers are not rendering pages and invoking actions, but are just considering the HTML source code. What big search engine providers like Google are doing, is to use their power and dominance to make web developer provide the content of their applications in a way possible to index it. The main idea is to make an AJAX application also accessible without using JavaScript by using simplified AJAX, i.e. to provide a static URL besides the JavaScript function call of hyperlink elements. When clicking these links a static page would be opened if client-side scripting is disabled. This way crawling is possible again.

[20] `http://www.sikuli.org`

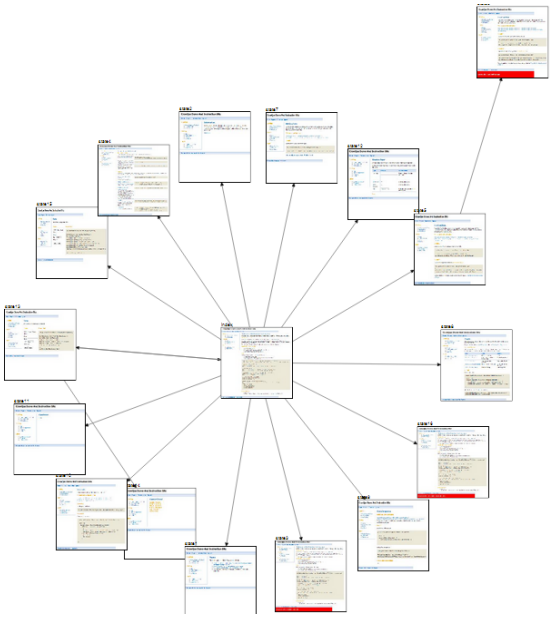

Figure 4.11: State-flow graph created with Crawljax

In [30] a so called state-flow graph is constructed by identifying and invoking all possible actions. An example is shown in figure 4.11. The identification of invokable elements is already the first non-trivial task which can only be performed if JavaScript is enabled because some parts of a web page may just get rendered at run-time. In their approach they limited the considered actions to clicks. The nodes of the created graph represent the state of the application, i.e. the DOM. The edges represent the clickable element which was invoked to change the state. Their algorithm is performed in a depth-first manner, meaning that as soon as the clickable elements on the page are identified the first one is invoked and the resulting page is again checked for actions to be invoked. In the end the state-flow graph is used to generate static pages. An important issue they pointed out, is that the navigation between the states can become very complicated, because if a recursive call in the depth-first algorithm is finished the browser state has to be changed to the state before. It was described that there are basically two solutions, the first one is to navigate to the desired state from the beginning. The second is to use the browser history support introduced with the History API in HTML5 or JavaScript workarounds.

The concepts described in their paper are furthermore implemented by their tool called Crawljax[21]. Other major issues of AJAX crawling were pointed out by [15]. One of them was that events can be very granular which can result in a large set of similar states. Another issue is that events may be invoked infinitely because the web page keeps changing. In [30] this issues were considered by defining a certain threshold when comparing two DOM states. In [28] these problems are handled by explicitly selecting the elements on a web page which are considered for the comparison of states.

[21] `http://crawljax.com`

Even though there are still problems to solve, these static pages, which can be generated using AJAX crawling tools, could directly be used as input for the repackaging process.

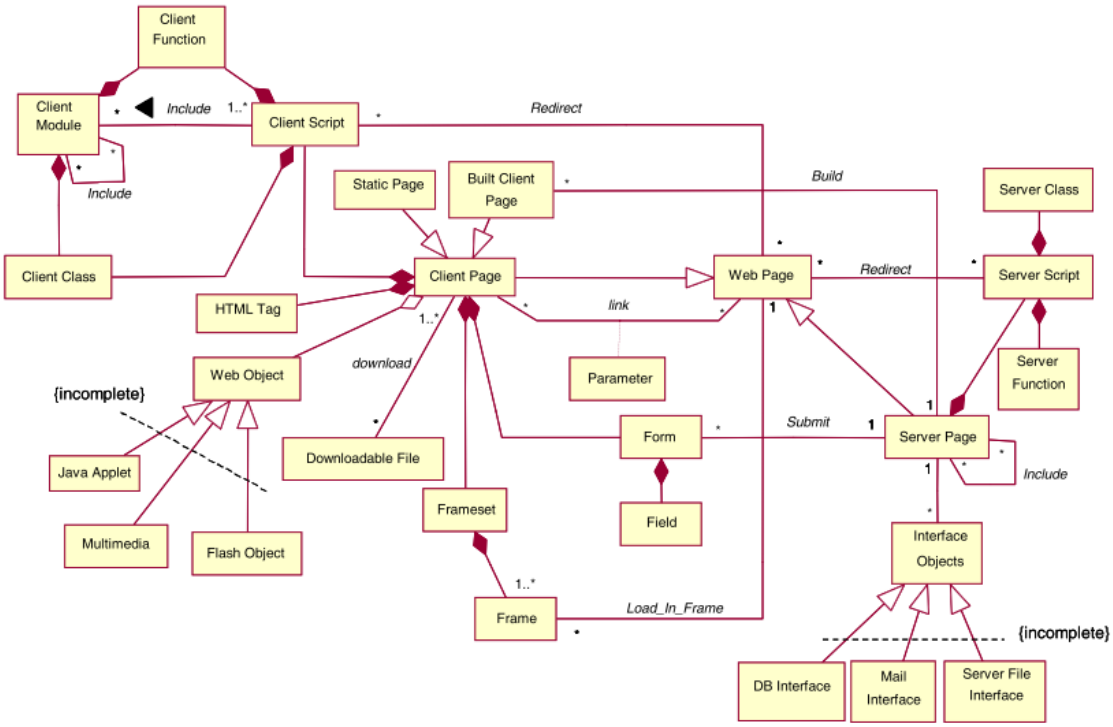

Figure 4.12: Reference model of a Web application. [14]

In [14] a web application is modelled from a reverse engineering perspective. They called their approach WARE (Web Application Reverse Engineering). The basic principle of reverse engineering is to identify components and their relationships to create an appropriate representation of the system on a higher abstraction level. For this purpose they extended an existing model for building web applications. This model is shown in figure 4.12. It is used to represent the entities of a web page as an UML[22] class diagram. Apart from that also the functional requirements and the interactions between entities are modelled in the WARE approach. To model the requirements UML use case diagrams were chosen. Interactions are represented using UML sequence diagrams.

[22]Unified Modeling Language - `http://www.uml.org`

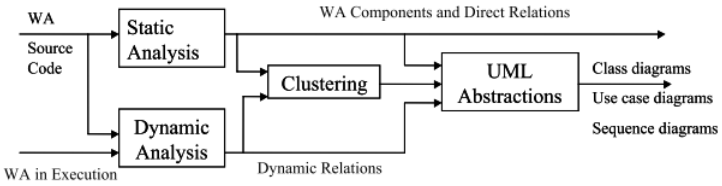

Figure 4.13: Reverse engineering process in WARE (WA means Web application) [14]

The detailed process of WARE is shown in figure 4.13. In the first step, the static analysis, the HTML code is analysed and the class diagram is instantiated and filled with the obtained static information. In the second step, the dynamic analysis, the dynamic behaviour of the application is observed and this new data is added to the class diagram. Within their analysis of the approach they found out that this step required intervention in order to define input values for forms or just to trigger a certain action. Based on the collected information clustering is performed. In this step the entities and their interconnections are searched for cohesive clusters, i.e. entities which are highly connected with each other. These clusters are then used in the next step, the UML abstractions, to create the use case diagrams. Every cluster may represent one use case. But it is mentioned that this reasoning just leads to suggestions and further manual effort to verify and annotate the use case diagrams will be required. In the end, for each use case diagram a UML sequence diagram is generated. Unfortunately some parts can not get fully automated, e.g. the cluster validation, which makes it not feasible for an on-the-fly repackaging process. But if those kinds of models would be part of the web development process the effort and costs to create web pages adapted for specific devices can be reduced. [14]

The result is much more granular than the resulting models of AJAX crawlers described before. This means it may not only be the input for repackaging, as it is the case with static web pages generated by AJAX crawlers, but it could be an essential part of the Transformation phase.

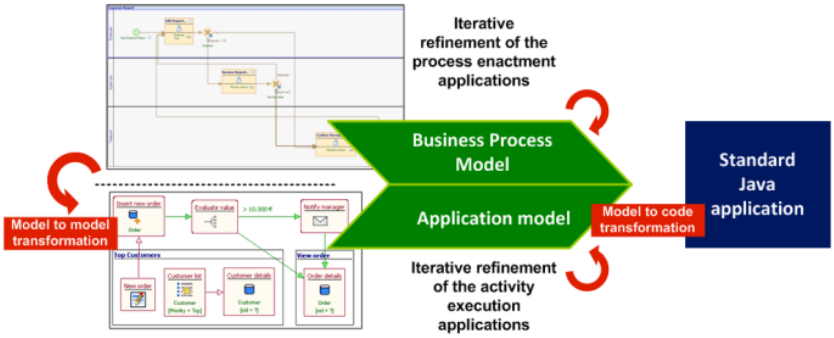

Figure 4.14: Transformation process in WebRatio [4]

Approaches where software is generated from models, are called Generative Software Development, Model-driven software development (MDSD), or Model-driven Engineering (MDE). These techniques have not yet realised their potential within the whole development cycle but are often used for models close to the implementation level, e.g. for transforming an ER diagram

into SQL code. A tool using the before mentioned IFML is WebRatio[23]. One major aspect is that it tries to separate the concerns process design and control, and user interaction. To describe the process BPMN[24] is used, and IFML is used in parallel to express the internal behaviour of each activity, e.g. back-end invocations or user-interface aspects. A first basic version of the IFML models is generated by the tool, i.e. one model for every activity in the process. This process is also shown in figure 4.14. In order to be flexible and extensible the tool uses a template-based approach which allows for example the replacement of user-interface parts or code-generation logic. The output of the tool is a web application covering the modelled business process. Possible ways to extract the business process out of a web page are discussed within the process transformation section 4.4.4. [4]

4.4 Transformation phase

4.4.1 Content transformation

After the content which has to be further processed is available in certain format the next step is to transform it into a different representation. The type of transformation can be distinguished depending on the source and the target format of the transformation. Especially interesting in this regard are code or model formats. Code-to-code transformation, as the name already states, means that input and output of the transformation are code, e.g. a conversion from Java code into HTML, or from HTML again to HTML. If on the other hand a Web Modelling approach was used earlier in the process, a model transformation might be the better approach. The output of this transformation could again be a model, e.g. in a different format, or code, e.g. from an UML model to HTML code.

Code-To-Code transformation

A first simple code-to-code transformation would be filtering, which means the original page is parsed and the content which is not desired to be in the output gets omitted. The advantage is that the original structure is preserved and existing styles are not modified. But at the same time this also represents a disadvantage if the target format or device is very different to the original, e.g. the styles and layout of a web page are unclear and confusing on a mobile device reducing the content will not change this effect.

If filtering is not enough the output can be created programmatically. To support this task template engines can be used. By doing so it is possible to define basic templates for target devices, and to just fill the template "gaps" with content. Examples for such template engines would be FreeMarker[25] or StringTemplate[26].

[23] `http://www.webratio.com`
[24] Business Process Modelling Notation - `http://www.bpmn.org`
[25] `http://freemarker.sourceforge.net`
[26] `http://www.stringtemplate.org`

Model-To-Code transformation

Model-to-code transformation uses a model as input and usually generates text which can afterwards be interpreted by a compiler or interpreter. According to [11] there are two major approaches in use, i.e. visitor-based and template-based approaches. The idea of the first one is to traverse through the internal structure of the model and output code in text format while doing so. Similar to the code-to-code generation, a template engine is used for the template-based approach. The engines will require that template files are written in a template language. This language allows to iterate and access the input model in order to fill the parts of the template with that data. An example for this would be the Eclipse Model to Text (M2T) project [27]. This kind of transformation process is for example used by WebRatio, where a web application is generated out of BPMN models. If the web page which is to be repackaged would be available in such a model or if such a model could be extracted out of the web page, such a model-to-code transformation technique could be applied.

Model-To-Model transformation

"Model-to-model transformations translate between source and target models, which can be instances of the same or different metamodels." [11] To accomplish this task rules have to be defined to decide how a model element in the source model is matched and converted to a model element in the target model. An example for a transformation language combined with an transformation engine for this purpose would be ATL[28].

Figure 4.15: ATL model-to-model transformation

4.4.2 Semantic Web

The main concept of the Semantic Web is to make content within the Web understandable by humans and machines. Therefore it is necessary to define the meaning, the semantics, of constructs like words or phrases.

> The Web was designed as an information space, with the goal that it should be useful not only for human-human communication, but also that machines would be able to participate and help. [2]

The biggest advantage of this "web of data" is that data is reusable and can be shared across web pages and application throughout the whole Web. Especially nowadays, where the amount of

[27] `http://www.eclipse.org/modeling/m2t/`
[28] ATL Transformation Language - `http://www.eclipse.org/atl/`

data available becomes overwhelmingly large, new ways to process large and complex amounts of data have to be found. To accomplish that, single entities have to develop from strings to things.

There are various ways to organize data with different semantic expressiveness. A taxonomy for example is a classification of things which is also capable of describing the relationship between concepts, e.g. parent-child relationship. If the associations between things of a class, e.g. synonyms or polysems, have to be represented, a thesaurus would be an applicable organizational schema. But the concept used for the Semantic Web are ontologies. It is not limited to hierarchical relationships between classes, it can relate any concept within the ontology with any other. As an example look at the Google search result for "Elisabeth of Austria"4.16. A taxonomy could may tell you that she was an austrian empress. In a thesaurus you may find that she was also called "Sissi", but only with an ontology it is possible to relate any two concepts by an attribute, e.g. where she was born, who her parents were, or how she died. An example for information retrieved from an ontology is also shown on the right side of the search result page. It is one of Google's approaches to the Semantic Web and is called Knowledge Graph[29].

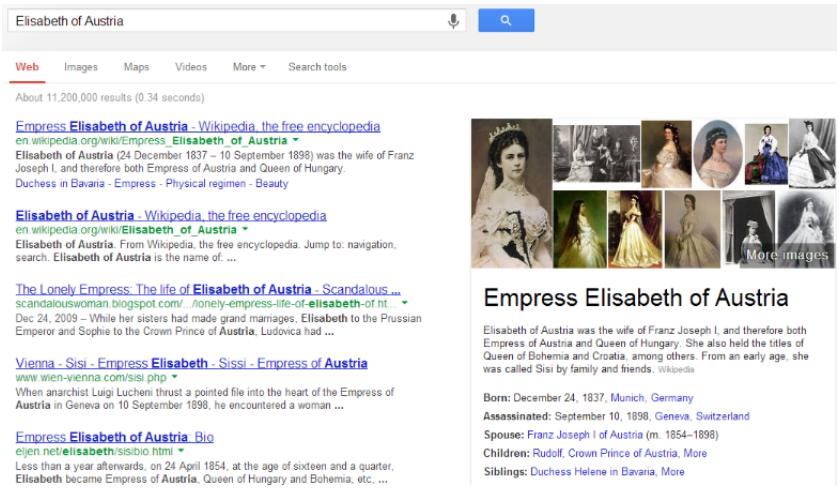

Figure 4.16: Google search result

The Semantic Web consists of web pages which made their content understandable. This can be accomplished by creating specific HTML tags or attributes with machine-readable information. A term which is used very often in this context is the one of Linked Data[30], which "refers to a set of best practices for publishing and connecting structured data on the Web" [3]. This makes it possible to link web content with information stored in ontologies or other collections of structured data. DBpedia[31] or Freebase[32] are just two examples of those knowledge repositories.

[29] http://www.google.com/insidesearch
[30] http://www.w3.org/standards/semanticweb/data
[31] http://dbpedia.org/
[32] http://www.freebase.com/

Within the transformation phase of the repackaging process parts of the information available in the Semantic Web could be integrated to further improve the utility of the repackaged content. For example references to popular literature found in a literature repository.

4.4.3 Web APIs

A web API defines an interface to a server accessible from the Web. These APIs can accept parameters and will return a response, usually in XML or JSON. And they are usually REST based (for REST see also section 3.2.2) . A related term is Mashup, which is a combination of several APIs and a combined representation of their results. Web APIs are offered by a lot of web application, especially Web 2.0 applications. Besides that, they are more and more replacing traditional Web Services. The advantage is that they are easier to use and also less complicated to develop. They can, for example easily be invoked using client-side scripting. A lot of companies do even provide wrapper components to make the usage of Web APIs more convenient, e.g. Facebook Javascript-SDK[33] or the Google Maps API [34]. An extensive catalog of web APIs can be found in the API directory of ProgrammableWeb[35].

The relevance of web APIs for the Transformation phase in the repackaging process is similar to the one of the Semantic Web. While the content is transformed it is possible to integrate a lot of additional information made available through web APIs. Scenarios reach from the integration of a map component for addresses to related videos or images. The difference to data from the Semantic Web is that the retrieved information has to be integrated in some way and can not just be referenced.

Figure 4.17: Programmable Web [26]

[33] http://developers.facebook.com/docs/reference/javascript/
[34] https://developers.google.com/maps/
[35] http://www.programmableweb.com

4.4.4 Process transformation

Process repackaging resp. transformation means not only to change the content of a single web page, but to adapt the whole process underlying several connected web pages. An example would be a purchasing process of an online shop consisting of multiple steps, e.g. going to the shopping cart, confirming the selected products, specifying an address, selecting a payment method. Assume such a purchase has single pages for every step, which are connected by "back" and "next" links. Process transformation would be a change within this whole process, e.g. to change the order of the steps or leave some out. The mentioned process could for example be converted into a one-click buy purchase, where just a single click is needed to complete the process if all the required data of the customer is already available. Content transformation on the contrary would be to adapt a single step, for example by optimizing it for a mobile device or adding some additional information like a warning if one of the products has a very bad rating.

The problem of recovering the structure of a web page was already discussed in section 4.3.2. In order to adapt the underlying business process of a multi-page web application it has to be available in a structured format. The easiest solution would be if the web page owner provides the business process. The problem is companies do often not have a documentation about their implemented processes, they are not in the right quality, or they are outdated. An approach introduced in [13] is to reverse engineer the process by observing user interactions. The result is a BPMN model. Pages belonging to a business process are assumed to have one or more *FORM* elements. These elements are observed while a user is manually performing the process steps. What is tracked during the execution is structural information, i.e. the visited web pages and their *FORM* elements including fields, and behavioural information, i.e. the sequence of submitted actions including data. Afterwards the process is dynamically extracted. Structural information is used to create activities within the resulting model, e.g. a select-payment form. The behavioural information is further used to create gateways, message and sequence flows. [13]

But what about web pages using client-side scripting to implement their processes without using multiple pages? In this case the structural information entities could be created artificially. For example one entity containing information about the current set of fields on the page, which gets created for each process step performed on the page. Also the behavioural information is manageable because additionally to form submits non-periodic AJAX actions could be seen as steps.

All this gives a lot of input to define how a process repackaging implementation could look like. Important is to get the underlying business process in a usable way. This could mean a BPMN model is already too abstract and some additional concept or model might be necessary, as IFML used within WebRatio [4] for example. The last step is then the actual transformation, where either model-to-model 4.4.1 or model-to-code 4.4.1 transformation approaches are feasible.

4.5 Existing tools and frameworks

Several repackaging tools are existing up to now. Some of them will be explained within this section. An example where not a web page but an RSS feed is used as input to create a mobile web

page would be Mippin[36] which is also shown in figure 4.18. It allows to specify the URL of an RSS feed and to adapt the basic style of the generated mobile web page. This includes for example the base colours, a logo image, or the title. Although there are a lot of mobile website builders available, e.g. fiddlefly[37] or goMobi[38], only a few support what can be considered repackaging because most of them require that the mobile web page is created completely manually.

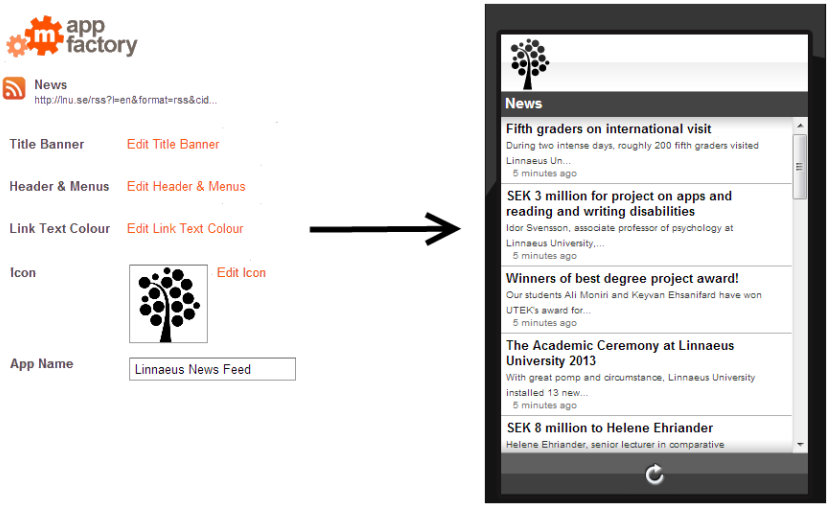

Figure 4.18: Transformation with RSS input and base style definition - mippin App Factory

One of the most advanced tools in the area targeting mobile devices is DudaMobile[39]. It does automatically extract the navigation and content HTML elements of a web page and adapts the CSS style to fit for mobile representation. They provide an editor to further change the basic style, e.g. the way the navigation menu looks like (as shown in figure 4.19). Feature-rich are also their live editing possibilities, which allow to add new elements like paragraphs, image galleries, or social plugins directly in the preview of the repackaged web page (an example is shown in figure 4.20). In the last step the web page is published to their web servers and can be accessed.

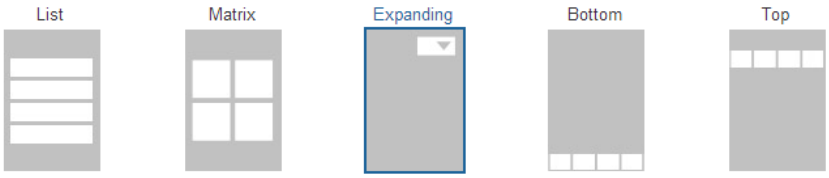

Figure 4.19: The different layout options of DudaMobile

[36] http://www.mippin.com
[37] http://www.fiddlefly.com
[38] http://gomobi.info
[39] http://www.dudamobile.com

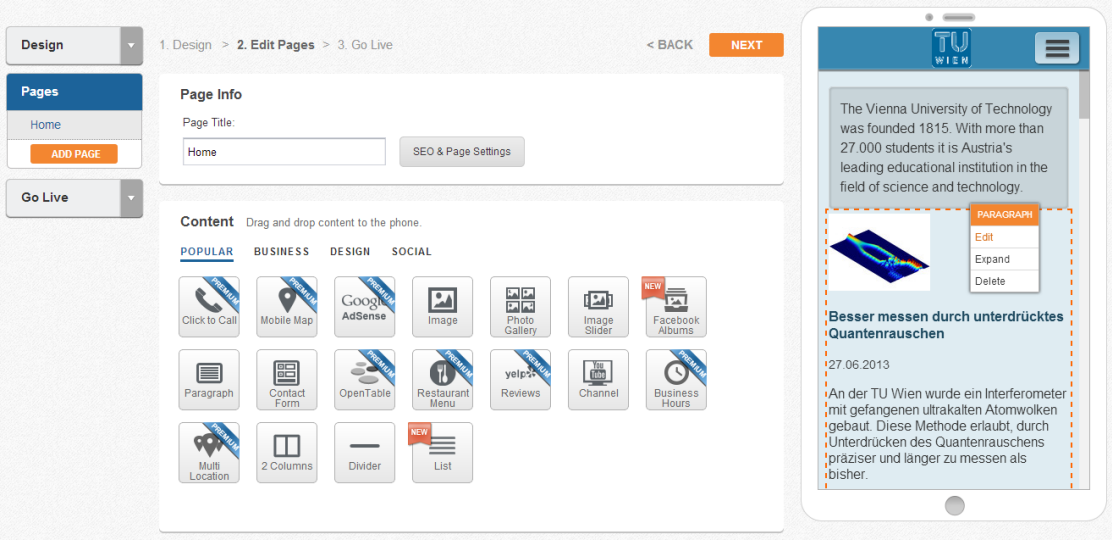

Figure 4.20: Editing the live preview of a web page transformed with DudaMobile

Another tool coming with automatic content extraction features is GinWiz[40]. It provides similar functionality like DudaMobile with the difference that it provides an advanced editor allowing to correct misclassification of content, e.g. if the wrong navigation bar was used in the transformed version. See figure 4.21 for example. Instead of the navigation bar on top the one on the left side should be used for the main navigation. This can be corrected within the editor. Another tool with a similar set of features which should be mentioned is bMobilized[41].

[40] http://www.ginwiz.com/
[41] http://www.bmobilized.com

Figure 4.21: Selecting content and correcting misclassifications with GinWiz

A shared problem of all the tested tools is that they work well with simple web pages, but as soon as the pages reach a certain complexity or are using AJAX technologies they struggle to achieve good results.

Another group of tools is those performing client-side repackaging. An example would be the Readability Chrome plugin which is a wrapper around the Readability platform[42]. This plugin allows to extract the main content of web articles and to represent them in a representation which is easier to read. To accomplish this, it uses a variety of metrics and heuristics on DOM level. An example is shown in figure 4.22. Just a part of the original article on the left side is content, and it is not even the full article, because there is a link to the second page shown. Nevertheless, Readability is able to extract that content as well. Another example would be the Chrome Daltonize![43] plugin which adapts images on a web page for colour-blind users. In general, a high percentage of web browser plugins are performing some kind of client-side repackaging, whether it is a translation plugin or an advertisement remover.

[42] `http://www.readability.com`
[43] `http://www.daltonize.org`

Figure 4.22: Conversion of a news article with Readability Chrome plugin

4.6 Future challenges

The development towards a web of APIs and Linked Data could allow to retrieve content in a standardized way and to make the Analysis phase of the repackaging process obsolete, because this APIs and data can be used to collect all the required information needed create a modified version of a certain web page. These APIs could also be directly integrated into models used for the creation generation of a web page.

The amount of data on the web is overwhelming and we struggle to make it accessible in an optimal way. The Linked Data concept is a logical consequence of this development. With the transformation of the Web as it is right now into a Semantic Web where data will in general be more open, also the need for web page segmentation will decrease. But on the other hand the possibilities within the Modelling and the Transformation phase of the repackaging process will become more exciting because of the easier combination of services and data.

CHAPTER 5

Repackaging

5.1 Client-side repackaging

Client-side repackaging is a method where the content of a web page gets modified for a specific purpose, e.g. to improve the user experience or the usability. Important is that the modification happens on client-side and that the browser does not simply show content generated on a server. The component performing the modification does furthermore not belong to the original provider of the web page, but is for example done by a browser plugin. A common example would be an advertisement-blocking plugin which evaluates a page as soon as it is loaded and removes content identified as advertisement.

The purpose of client-side repackaging within the MVT project is to replace the generation of heatmaps as described in section 3.5.4 by drawing them directly within the currently opened web page. The advantage is that the generation of screenshots is not necessary anymore, and the overlaying of the heatmap does not have to be done on server-side, but directly on the client, which is computationally faster because the web page is already loaded.

A basic design decision which has to be taken to enable client-side repackaging is how the content can be injected into web pages. The approach taken in [45] is to use a browser plugin which can access and modify web pages. The approach used within this thesis injects additional information using a proxy which has to be configured for the browser in use. This is shown in figure 5.1. The script builds upon the client-side scripting component of the MVT project 3.5.1 and adds its own functionality.

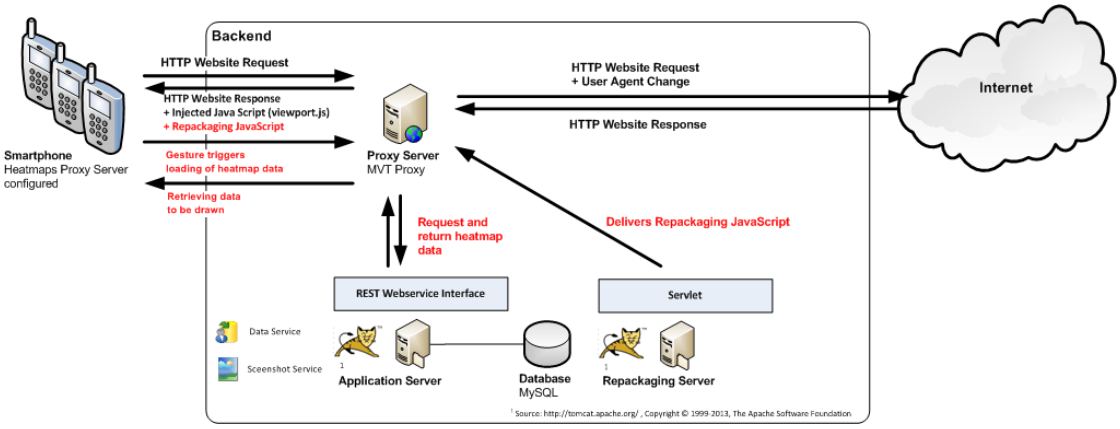

Figure 5.1: Client-side repackaging architecture

The script is initialized as soon as the web page is loaded. Within this initialization Hammer.js is used to register a gesture, i.e. swiping to the right, down and then left within a certain time. Besides that a third party JavaScript library supporting the drawing of heatmaps within a web page called Heatmap.js[1] is loaded. If the gesture is triggered the heatmap data is retrieved from the server and drawn. As mentioned earlier, this can not be done in a straight-forward way because of the same origin policy of web browsers, which prevents that data is retrieved from a web server which is not the host server of the web page. The heatmap data is therefore retrieved using a technique called JSONP.

Listing 5.1: Dynamic script loading

```
1 MVT.loadScript = function(scriptURL, loadCallback) {
2         var script = document.createElement('script');
3         script.type = 'text/javascript';
4         script.async = true;
5         script.src = scriptURL;
6         var head = document.getElementsByTagName('head')[0];
7         script.onload = loadCallback;
8         head.appendChild(script);
9 };
```

Listing 5.2: Call to retrieve heatmap data

```
1 MVT.loadScript("http://host/MVTService/rs/dataCollector/importantContent?callback=MVT.
      Repackaging.heatmapDataLoaded&th=5&url="+
2 document.URL);
```

The basis to load the data using JSONP is the method *MVT.loadScript* 5.1 which dynamically creates a new *script* element and adds it to the *head* element of the web page. The URL for loading heatmap data by invoking the REST web service is shown in 5.2. Important are especially the parameters. The *url* parameter is necessary to find the current heatmap for this web page.

[1] https://github.com/pa7/heatmap.js

The *th* parameter defines the threshold resp. the lower bound for the heatmap values. A value of 5 means for example that heatmap cells are only included in the result if their value is higher than 5 percent of the overall maximum value. This can be used to reduce the amount of data transferred. The last parameter, *callback*, is the name of the method which is called with the heatmap data as parameter. This method has to be an already existing JavaScript function, e.g. *MVT.Repackaging.heatmapDataLoaded* as shown in the example. This is also how the same origin policy gets bypassed, because *script* elements referencing other domains can be integrated, where on the other hand AJAX calls will never retrieve a response. The retrieved heatmap data is basically an array of cells where important content was identified including their position and actual value. In the method handling the result data a new Heatmap.js instance is created and the data is adapted, inserted, and drawn. Figure 5.2 shows a comparison of a heatmap generated on server side and a screenshot of a heatmap generated on client side using the described approach. The minor differences are based on the different drawing algorithms.

Figure 5.2: Comparison of client-side (left) and server-side (right) heatmap creation

5.2 Server-side repackaging

In contrast to client-side repackaging, the server-side pendant is not limited to the functionality provided by a client, but can make use of more complex algorithms or frameworks running on a server. The basic principle stays the same, the original version of a web page is considered as input, afterwards the desired phases of the repackaging process, as described in chapter 4, are performed, outputting a new version of the web page.

The goal within the MVT project is to evaluate how the collected data about user behaviour can be used to enhance the repackaging process. The idea is that it should be possible to generate a mobile version of a web page which focuses on content identified as important by other users. Therefore an on-the-fly solution running on a Java web server was developed.

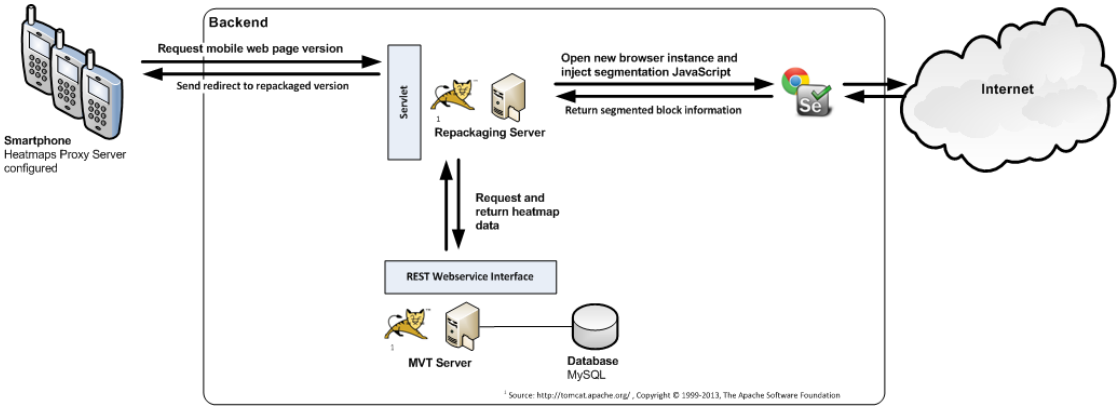

Figure 5.3: Server-side repackaging architecture

The basic architecture is shown in figure 5.3. To retrieve the repackaged version of a web page it has to be requested from the repackaging server. In this case the URL will be like `http://host/MVTRepackage/mobile?page=http://www.webpage.com`. Afterwards the various steps of the repackaging process are performed, the transformed web page is published on the web server, and the client automatically gets redirected to that page. In the following sections the implementations of the different phases of the process are described.

5.2.1 Content segmentation

After an analysis of existing web page segmentation techniques (see section 4.2.3), a feasible approach for this repackaging process has to be found. Therefore the following criteria have to be evaluated for their fulfillment:

- *Output*
  It is not enough to only use text for the further processing in the following phases. The underlying HTML has to be available to perform specific content selection and to have more extensive possibilities for the content classification. Besides that the final output will also be HTML, which makes it desirable to keep the format through the process.

- *Integration*
  It has to be possible to integrate the algorithm or framework into another code base resp. project and not having the output in a separate tool only. This need is based on the design decisions of the architecture, i.e. to use a Java web server which delivers the output.

- *Quality*
  The chosen technique has to perform a segmentation in a reliable, accurate, and faultless way.

Based on these requirements only one approach was found applicable. The problem of XY-Cut is that it takes a document image as input and outputs the identified segments without information about the HTML elements within that region and how they are structured. The disadvantage of the WPPS is the integration because it is implemented as an independent tool. The Block Fusion implementation boilerpipe alone can also not be used. Although it supports the output of HTML it would always just return the main content of a web page. But it proofed to be useful in other steps of the repackaging process, i.e. the content selection. VIPS fulfills all the requirements, the input as well as the output is in HTML format with additional information. The integration is not a problem because the algorithm can be implemented in any environment with access to the HTML source. The quality of the results was already verified within other works like [8].

VIPS-based segmentation

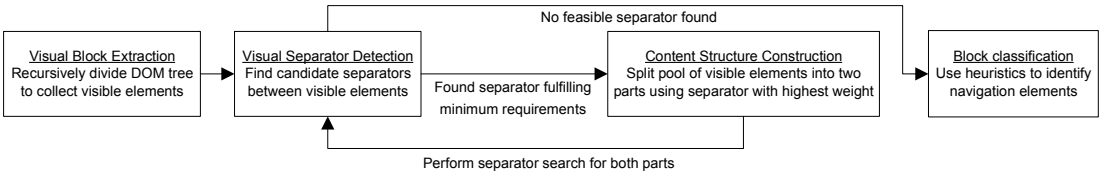

Figure 5.4: VIPS-based approach

The algorithm to segment a web page is implemented within JavaScript and is based on VIPS, which is described in section 4.2.3. Before this approach was used a first prototype tried to perform the segmentation within Java using Selenium to load the HTML elements which are needed, but unfortunately a lot of attributes of elements are not easily retrievable without an additional call to the browser instance. As a result a large amount of data was transferred all the time and the performance in general was also not acceptable.

An overview of the resulting approach is shown in figure 5.4. The first step, called Visual Block Extraction, is a recursive dividing of the DOM tree to find the most granular visual elements. It starts with the *document.body* as input. The corresponding method is shown in listing 5.3. If the *node* parameter is an element which can be divided further, e.g. if it contains a lot of visible child elements the *divideDOMTree* method is called for all child elements. If the element is visible and granular enough it is added to the list of visible elements, otherwise it may be ignored resp. cut.

Listing 5.3: Recursively dividing the DOM tree

```
divideDOMTree : function(node, level, pool) {
        var nResult = this.isDividable(node, level);
        if(nResult == this.DIVIDE) {
                var children = node.childNodes;
                for( var i = 0 ; i < children.length; i++ ) {
                        this.divideDOMTree(children[i], (level+1), pool);
                }
        }
        else if (nResult == this.CUT) {
                this.cutBlocks.push(node);
        }
        else if(nResult >= this.VISIBLE) {
                this.assignDoC(node, nResult);
                pool.push(node);
        }
}
```

In order to decide if a certain HTML can be divided heuristic rules are used. This happens within the *isDividable* method. The set of heuristic rules is based on the rules described for VIPS [8]. These rules are responsible to decide whether a certain HTML element has to be further divided, can be cut, or is considered a visible element. A detailed list is shown in 5.2.1. For the description of the rules the following definitions are used:

- *Text node:* a DOM node of type text, i.e. elements with *nodeType* of 3 which do not have an own tag name but do just contain text
- *Virtual text node:* a DOM node which just contains text nodes or other virtual text nodes
- *Content container:* a DOM node of type link, paragraph, or headline. These special containers are often more usable if they are not further divided, e.g. a link just containing a virtual text node
- *Content node:* a DOM node which can either be a text node, a virtual text node, multimedia content like *IMG* or *EMBED*, or a content container which just contains text
- *Valid node:* a DOM node which is visible on the web page, i.e. has no negative offset or visibility attribute set to be hidden

Table 5.1: Heuristic rules for dividing blocks

Rule 1	If the DOM node is not visible and has no valid children it will be cut
Rule 2	If the DOM node is not a content node and it has no valid children, then this node cannot be divided and will be cut as well
Rule 3	Form elements are always added, even if invisible
Rule 4	If the DOM node is a content node without valid children and a size smaller than a threshold, it is also cut
Rule 5	If the DOM node has only one valid child and the child is not a content node, then divide this node
Rule 6	If the DOM node is the root node of the (sub-)DOM tree, divide this node
Rule 7	If all of the child nodes of the DOM node are text nodes or virtual text nodes, do not divide the node
Rule 8	If one of the child nodes of the DOM node is and *HR* element, then divide this DOM node
Rule 9	If the DOM node is a content container element containing only one visible child, do not divide the node
Rule 10	If the sum of all the child nodes' size is greater than this DOM node's size, then divide this node
Rule 11	If the background colour of this node is different from one of its children's, divide this node
Rule 12	Do not divide DOM nodes of type *TR* or *TD*
Rule 13	Divide all other DOM nodes

The next step is the Visual Separator Detection. The steps described here are taken from [8]. At the beginning the two lists for vertical and horizontal separators are initialized with one separator in each of them. These separators span the whole block, i.e. all visible elements. For both lists, all the visible elements in the block are compared with all the separators in the corresponding separator list. This leads to three different possibilities for each comparison:

1. If the visible element is contained in the separator, the separator is replaced by two new separators at the beginning and end of the element
2. If the visible element overlaps the separator from one side, the separator size and its weight are updated
3. If the visible element covers the separator, the separator is removed

In the end the four separators at the borders of the list are removed because if they would be chosen the block would not be split.

An essential step during the creation of new separators is the assigning of a certain weight to them. This weight is used to determine which separator will be chosen for the Content Structure Construction. The following aspects determine this weight:

- The bigger the distance between the start and the end position of the separator the higher the weight
- If background colours of the elements on the two sides of the separator are different, the weight will be increased
- If the font size and font weight of the elements on the two sides of the separator are different, the weight will be increased
- If there is a border between the elements on the two sides of the separator, the weight will be increased

Figure 5.5: Different threshold value results in more granular (right) or coarse (left) segmentation

A difference compared to VIPS is that there is no granularity value calculated for every visible element, but just the separator weights determine if the structure will be further divided. This means the Visual Block Extraction is only done once and depending on a configurable threshold for the minimum separator weight the segmentation will be more coarse or granular. Figure 5.5 shows the resulting segmentation with a different threshold value. An important aspect is that the visual blocks contained within these pools always have the same granularity.

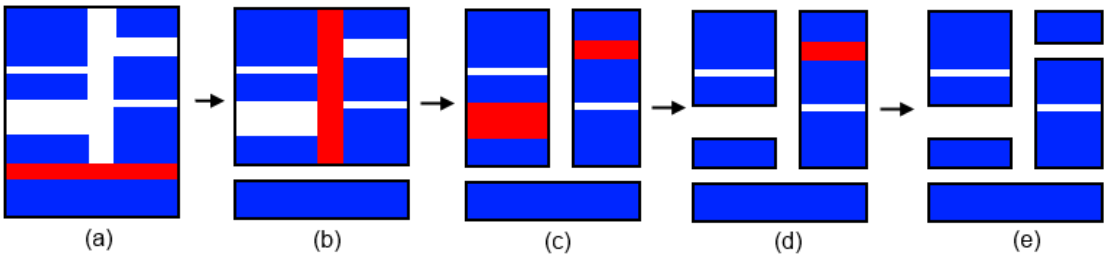

Figure 5.6: Content Structure Construction example

The Content Structure Construction is also different to VIPS because there is no hierarchy created, but a list of blocks containing one or more visible elements. After the separators were identified the separator with the highest weight is chosen and the current pool of visible elements is split in two parts, the elements residing on the one side of the separator and the elements on the other side. The original pool is removed afterwards. For both pools the previous steps beginning with the Visual Separator Detection are performed again. Figure 5.6 shows an example. The black borders represent the pools, blue elements are visible blocks, and red indicates possible separators. The bigger the width resp. the height of the separator the bigger the weight. In the beginning (a) only one possible separator exists which results in two new pools for which new separators are identified, as shown in (b). This process continues until the separators do not fulfill the minimum requirements anymore, as shown in (e).

The result of the Content Structure Construction is a list of pools with with a certain dimension and a list of visible elements. Another difference to VIPS is that the Content Structure Construction is not performed in a bottom-up but top-down way, i.e. it starts by chosing the separator with the highest weight instead or merging the ones with the lowest weight. For demonstration purposes the script is also able to draw the identified pools and elements directly within the browser. An example for this is shown in figure 5.7.

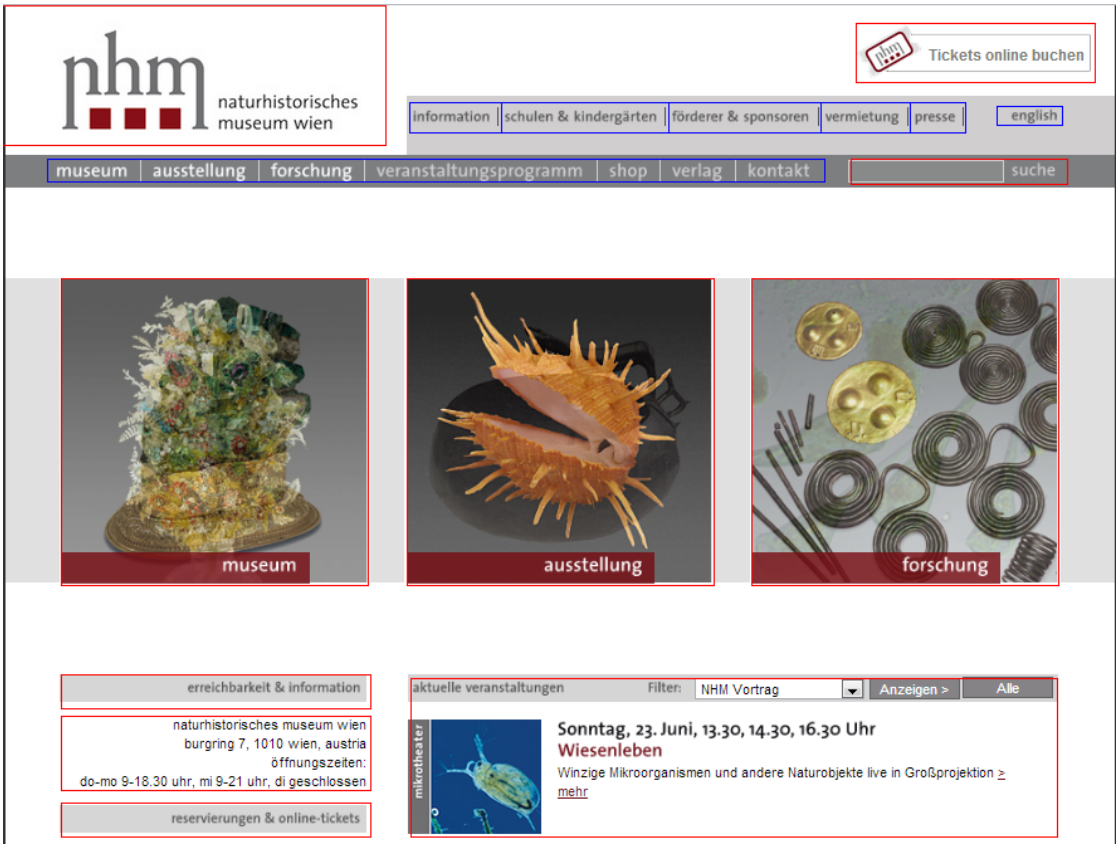

Figure 5.7: Visualized example of the resulting segmentation

The segmentation is performed completely in JavaScript which makes it necessary to load the page which has to be repackaged, execute the JavaScript and return the result to the web server performing further actions. Therefore Selenium is used. It opens a new browser instance and goes the desired page. Afterwards Selenium injects the JavaScript file containing the code to perform the segmentation. Selenium periodically checks if the segmentation was finished, and if that is the case it retrieves the identified *pools*. For every *pool* a *Block* object with the following information is created:

- *Size:* Information about X and Y coordinates, and about the width and height of the block
- *Text:* The text content of the block. This text is the combined text of all the HTML elements within the block
- *Text density:* As soon as the text of the whole block is known its density can be calculated. Therefore the formula described within 4.2.3 is used for the implementation shown in 5.4

Listing 5.4: Calculation of the text density

```
1  public float calcTextDensity(String text)
2  {
3          String[] lines = text.split("\n");
4          StringBuilder sentence = null;
5          int lineNumbers = 0;
6
7          for (String line : lines)
8          {
9              lineNumbers++;
10             sentence = new StringBuilder();
11             String[] words = line.split(" ");
12
13             for (String word : words)
14             {
15                 sentence = sentence.append(word);
16                 // Returns the width of the text in a predefined font
17                 int sentenceWidth = FontMetricsUtil.getInstance().getWidth(sentence.
                       toString());
18                 // If the length exceeds the line width constant
19                 if (sentenceWidth > WIDTH)
20                 {
21                     lineNumbers++;
22                     sentence = new StringBuilder();
23                 }
24             }
25         }
26         return text.replaceAll("\n", " ").length() / lineNumbers;
27 }
```

- *Elements:* List of visual HTML elements within the block
- *Container:* The container HTML element does not correspond to the parent element of the block but contains a reference to a structural parent for the block, e.g. a *FORM* element which can not just get omitted because no action would be performed if only the child elements of the *FORM* would be repackaged

- *Class* The content classification is done on client-side as well. Depending on certain heuristics the class of the block is defined, e.g. ”navigation”

5.2.2 Content classification

The next step in the process is the classification of content. For this solution, the content was divided into two classes, “navigation” and “content”.

The classification for the navigation is based on heuristics. The goal was to identify those blocks which contain visible elements used for navigation, e.g. toolbar entries. The first part of the classification is independent of the segmentation. It searches for navigation element candidates within the whole web page. Therefore a list of possible HTML element types was defined, which includes *DIV* and *UL*. Besides that a list of common keywords for navigation elements was defined. This list contains words like “nav”, “menu”, or “options”. The decision if an element within the web page is a feasible candidate is depending on a matching tag name and one of the keywords has to exist within either the *role*, *id*, or *class* attribute. The navigation element candidates are very often the containers of toolbar entries but do usually not contain single navigation elements.

Listing 5.5: Pool classification

```
classifyPools : function() {
        var navCandidates = this.getNavCandidates();

        for(var i = 0; i < this.pools.length; i++) {
                var pool = this.pools[i];

                for(var j = 0; j < navCandidates.length; j++) {
                        var nContained = 0;
                        for(var y = 0; y < pool.els.length; y++) {
                                if(this.isDescendant(navCandidates[j], pool.els[y])) {
                                        nContained++;
                                }
                        }
                        if((nContained / pool.els.length) > 0.8) {
                                pool.cls = this.CLS_NAVIGATION;
                                pool.container = navCandidates[j];
                                break;
                        }
                }
                if(!pool.cls) {
                        pool.cls = this.CLS_OTHER;
                }
        }
}
```

The next step is to check if the visible elements within a pool lie within a navigation element candidate. This requires that the segmentation has been completed. Because the visible elements are usually more granular than the navigation element candidates it is checked if a certain percentage of elements in the pool belong to a navigation element candidate. If that is the case the pool is marked accordingly. This is also shown in listing 5.5. The classification is directly done after the segmentation but before anything is transferred to the web server. It is therefore returned to the web server as part of the pool information.

Navigational elements are already classified at this point. The next step is to identify content blocks. Therefore the Block Fusion algorithm (see section 4.2.3) is used to combine visual elements and to decide whether they are content or not.

Block Fusion

A key aspect of the content classification solution is the Block Fusion algorithm [23]. The implemented component takes as input the visual elements found within the content segmentation phase (see section 5.2.1). For convenience reasons, the text density of each element was directly calculated when the corresponding Java *Block* object was created. When performing the algorithm, every block is compared with its neighbours. Every comparison results in one of the following two cases:

1. If the difference between the current block and the following block is lower than a certain threshold, both blocks are merged to a new one replacing the two old blocks. This new block has a new text density and a reference to the HTML elements of both original blocks

2. If the difference between the predecessor and successor of the current block is very low and both have a text density which is higher than the one of the current block, all three blocks are merged to one single block. Similar to the previous case this new block has a new text density and references to all relevant HTML elements.

The blocks are iterated multiple times, i.e. as long as a block was merged with another one. The method is shown in listing 5.6.

Listing 5.6: Block

```
while (merge) {
    merge = false;
    List<Block> blocksToMerge = new ArrayList<Block>(blocks);
    blocks.clear();

    Block prevBlock = null, nextBlock = null, block = null;
    for (int i = 0; i < blocksToMerge.size(); i++) {
        block = blocksToMerge.get(i);
        nextBlock = blocksToMerge.get(i + 1);

        // Merge with next block
        if (getDelta(block, nextBlock) < 0.5) {
            Block aBFBlock = this.createNewBlock(block, nextBlock);
            blocks.add(aBFBlock);
            i++; prevBlock = null; merge = true;
        }
        // Merge block with both neighbors
        else if (prevBlock != null && getDelta(prevBlock, nextBlock) < 0.15
                && nextBlock.getTextDensity() > block.getTextDensity()
                && prevBlock.getTextDensity() > block.getTextDensity()) {
            Block aBFBlock = this.createNewBlock(prevBlock, block, nextBlock);
            // Removing the previous block which is already in the list
            blocks.remove(prevBlock);
            blocks.add(aBFBlock);
            i++; prevBlock = null; merge = true;
        }
```

```
27          else {
28              prevBlock = block;
29              blocks.add(block);
30          }
31      }
32  }
```

5.2.3 Content selection

The next step is the selection of the classified elements. This means, from all the blocks which were identified using the segmentation the ones which are desirable for the result have to be chosen. The content was already divided into two classes, i.e. "navigation" and "content". Navigational elements are all selected for the transformation step, even if several navigation toolbars may be used. The content block selection iterates through the resulting elements of the Block Fusion implementation (see section 5.2.2) and uses those which have a text density higher than the configured threshold.

Important content

A more advanced approach is applied after the initial selection of the content elements. It checks if the chosen blocks are important enough to be transformed. The significance of the content is determined using user behaviour data collected with *MVT*. This heatmap data is available using a Web Service method. It allows to retrieve the importance matrix of the heatmap. For further details see section 3.5.3. A content block gets selected for the transformation, if there exists a heatmap cell within the dimensions of the block having a weight higher than a certain threshold. This threshold is a relative heatmap value, i.e. relative to the maximum weight within the heatmap, and is set to 25 by default. The impact on the result is further shown and analysed within the evaluation chapter 6.

5.2.4 Structure and content representation

The whole structure and content is represented within Java objects. If the web server hosting the repackaging solution is invoked to repackage a certain web page a new instance of a *ProcessedWebPage* is created. It has to be initialized with an implementation responsible for the segmentation, i.e. a Java class extending *ISegmentation*, and a class handling the classification, i.e. an extension of *IClassification*. The sections described above are concrete extensions of these interfaces. All the extracted, segmented, and classified HTML elements are available through their interfaces, see listing 5.7.

Listing 5.7: Interfaces of the internal representation

```
interface IProcessedWebPage {
    void process();
    IClassification getClassification();
    ISegmentation getSegmentation();
}

interface IClassification {
    void classify(String url, ISegmentation segmentation);
    List<NavigationEl> getNavigationElements();
    List<Block> getContentElements();
}

interface ISegmentation {
    void segment(String url);
    List<Block> getSegmentedBlocks();
}
```

5.2.5 Repackaging content for mobile devices

The transformation process of this implementation targets mobile devices. This means the final output has to fulfill several requirements to be nicely rendered on those devices. It should just use HTML and JavaScript to display content and should avoid technologies which are not supported, e.g. Flash. But the most important issue is that the size of the elements has to be adapted for the resolution of those devices and it should not be necessary to use horizontal scrolling for the navigation within the repackaged web page.

What is happening in this last phase is that the internal structure in form of a *ProcessedWebPage* is transformed into HTML and JavaScript code. Therefore the template engine FreeMarker is used. A template file containing a basic skeleton of a mobile web page was created. It uses jQuery Mobile[2] to create a smartphone-optimized navigation menu. The inputs for this template are basically two lists, one containing the content elements and another one for the navigation elements.

Important is to convert the data into a usable format for inserting it into the template. Therefore content elements are getting cleaned. A single content element may not just contain a text value but may have several HTML elements as children. For every element all attributes which are not relevant or would not have an effect in the mobile version are removed within the cleaning process. This includes for example *class* or *style* attributes. Other attributes, which are important for their corresponding HTML elements, are explicitly checked if they exist. This includes for example a *href* for links, an *action* for *FORM* elements, or a *src* for images.

Also the navigation elements are converted. Internally there are two types of navigational elements, links and groups. Every navigation group on top level may represent one navigation container, e.g. a toolbar. Web pages do often contain several of those top level toolbars, e.g. one or more top toolbars and a footer toolbar. These navigation groups may again contain navigation groups, e.g. for sub menus, or navigation links. This whole hierarchical structure is directly converted into navigation menu entries.

[2] `http://jquerymobile.com`

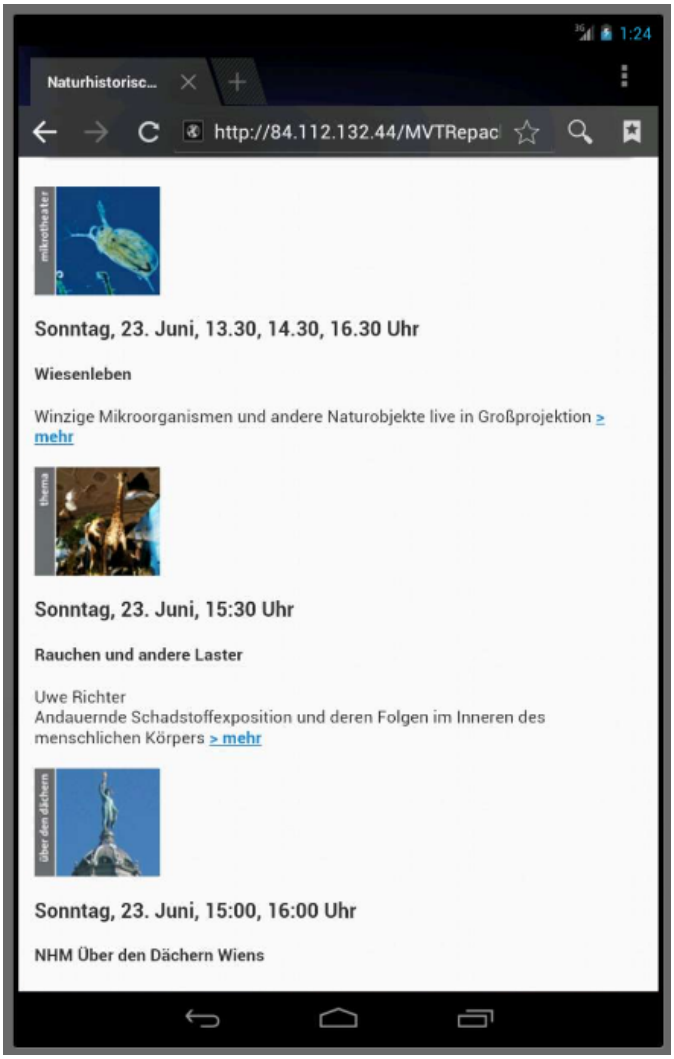

Figure 5.8: Repackaged mobile version

CHAPTER 6

Evaluation

In this chapter a closer look is thrown on the utility of the implemented solutions. The design evaluation methods which are used were already defined in section 1.5. Controlled experiments are used to demonstrate the functionality using web pages of two different domains, namely news and museums. Besides that, scenarios are used to describe advantages, disadvantages, limitations, and possibilities. In this part the artefacts are also compared to existing tools and frameworks 4.5.

The first step to reflect upon the implemented solutions is to evaluate the fulfillment of the research questions defined in section 1.2, which are based on unsolved problems by the environment.

How can repackaging of a web page for a specific target device be achieved? How could a feasible model / process look like? The process introduced within the state-of-the-art chapter 4 describes one possible approach. It tries to integrate all the models, tools, algorithms and concepts which were found to be essential for a comprehensive repackaging process. The repackaging solution created for this thesis shows that this model led to a repackaging solution capable to transform desktop web pages into web pages optimized for mobile devices.

How could an appropriate architecture look like? Within the state-of-the-art chapter a lot of existing tools and frameworks were evaluated to see what kind of architecture might lead to good solutions. In the end these solutions can be classified into two categories, client-side and server-side repackaging. For both of them artefacts were created and their resulting architecture is described within the repackaging solution chapter (see 5.1 for the client-side repackaging architecture and 5.2 for server-side).

How can the content used for repackaging be determined? This problem was ultimately discussed within the content selection phase of the repackaging process (see section 4.3.1). Basically, it can be said that the decision about which content has to be transformed often requires manual effort, especially because it is usually not possible to decide which content is important for a specific target device or format. Besides that, it depends on how good and reliable the previous step, the classification, was. In the implemented solution an algorithm measuring the text density was used

to decide if elements can be considered content. Here a manual approach allowing to explicitly select the content blocks used for the transformation would be possible as well.

How can the output of user behaviour analysis be used to support it? Within the implemented solution it turned out that the data collected using user behaviour analysis is valuable information for the repackaging processes. It allows to more accurately determine which content to select within the content selection phase, i.e. only content which was identified as important using *MVT* was selected for repackaging. This is also further discussed in the scenarios and controlled experiments section 6.1.

Can the captured web content be modelled and further processed? This question touches the Modelling and the Transformation phase of the repackaging process. Various concepts about Web Modelling are discussed in an own section 4.3.2. Also tools and standards for modelling web applications are discussed. This topic is also very much influenced by web UI testing and AJAX crawling because both require that a model of the web application is created in order to test or index it. Therefore also reverse engineering approaches are discussed. The possible ways of processing the models is then explained in section 4.4.1, i.e. the different ways of transforming a model either into code or another model.

Unfortunately there is no approach capable of performing this in a completely automated way, but they show promising concepts and full automation is also not necessary. There are applicable standards and tools emerging and as the complexity of web applications will most likely not decrease, the modelling of web content, including underlying business processes or not, may become more relevant in the future.

6.1 Scenarios and controlled experiments

The two domains chosen for controlled experiments are news pages and museum pages. The first is good for the comparison with other techniques and tools, and the second is good to illustrate the advantages of this approach. News pages mainly have a main page with a list of recent articles, and single pages for every article. Museum pages on the other hand provide all kinds of heterogeneous information, e.g. opening times, descriptions of exhibitions, ticket prices, current research, etc. In this case it is harder to determine what content should be used for repackaging.

First the overview pages of two news portals were repackaged. They are shown in figures 6.1 and 6.2. Depending on the threshold value used for the segmentation, pools are more granular or coarse (described in section 5.2.1). The visual blocks contained within the pools always have the same granularity. For simpler pages like the one of news.orf.at it is less of an issue because the whole list of articles can easily be converted. The problem becomes more obvious when looking at pages with a more complex layout, like the one of the Guardian. In that case a minor adaption causes changes in the resulting segmentation. But because in a later phase the Block Fusion implementation merges the article blocks again, as they have a similar text density, the problem becomes less important. Nevertheless the algorithm would be more efficient if the pools are correct because otherwise blocks could be wrongly merged. As a future task the heuristics and the threshold could therefore be improved.

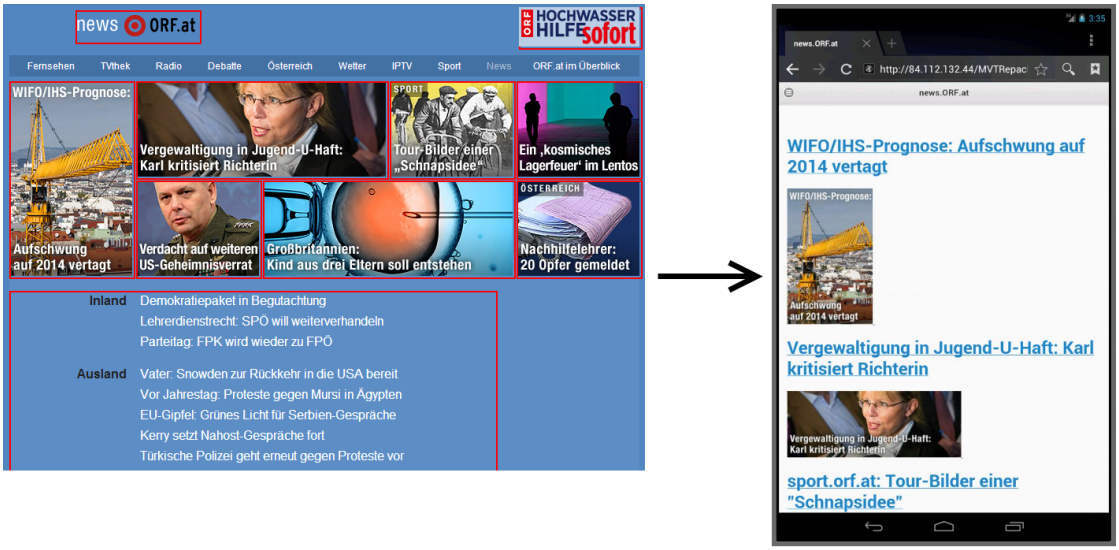

Figure 6.1: Repackaging of the news.ORF.at[1]entry page

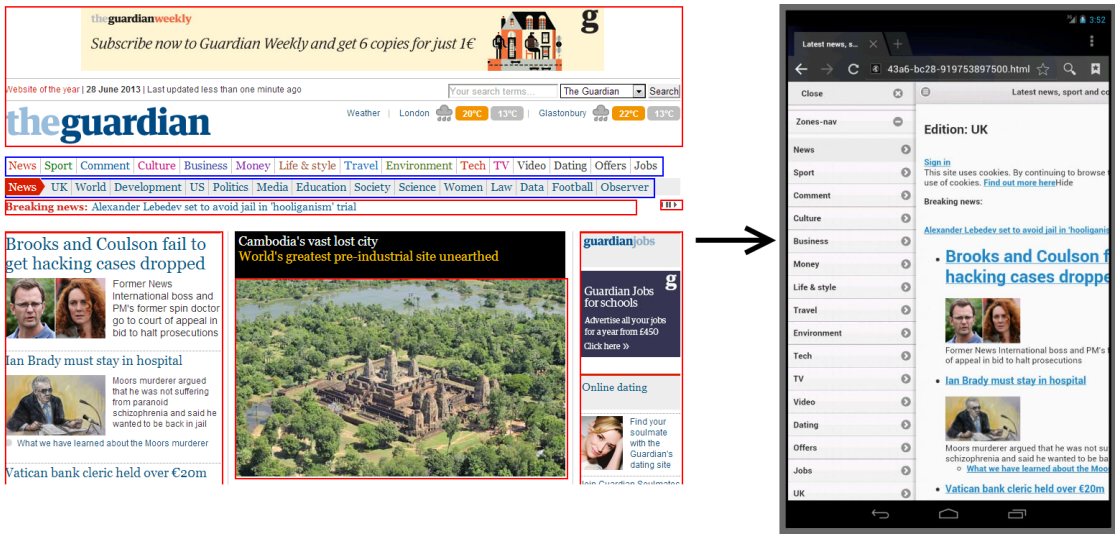

Figure 6.2: Repackaging of the Guardian[2]entry page

Another issue was the extraction of navigation information. The heuristics approach used to classify blocks as navigation showed to be successful in most cases. An example is shown in figure 6.2. The entries in the menu shown in the repackaged version on the right side are taken from the toolbars of the original web page (marked with blue borders). The problem of multiple toolbars is solved by creating a virtual group for each of them in the generated navigation menu.

[1] `http://orf.at/`
[2] `http://www.guardian.co.uk/`

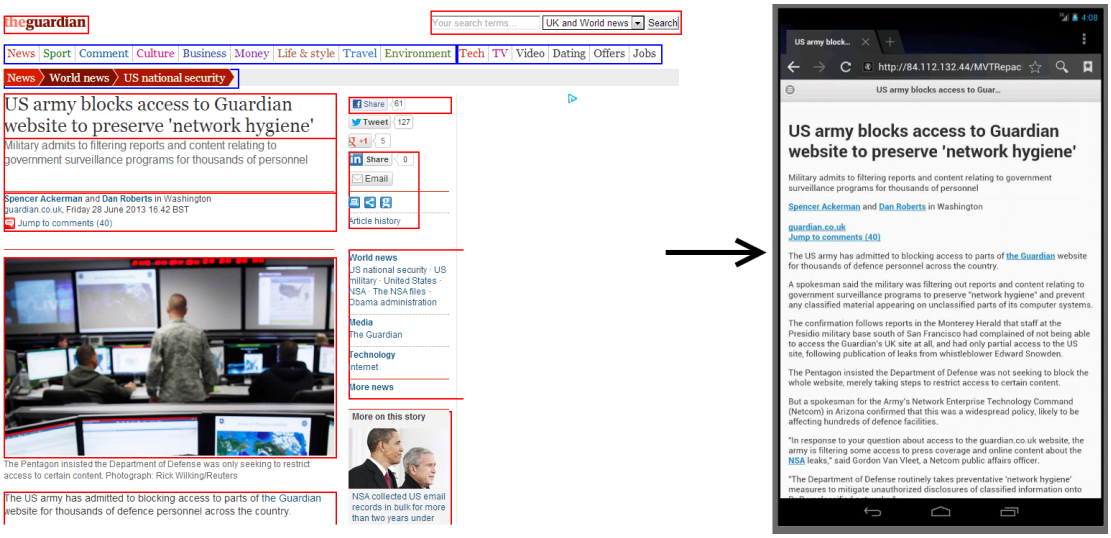

Figure 6.3: Repackaging an article page of the Guardian

The next scenario to be discussed, is when a web page contains one or several big blocks of content, e.g. an article page. In this case the implemented content classification, which is described in section 5.2.3, can easily find the main content of the web page. An example is shown in figure 6.3. Unfortunately several existing tools capable of extracting the main content of a web page are already available (see section 4.5). A big drawback of the whole approach within this first domain is also that most of the news pages do already provide a mobile version to view their content.

Where the implemented solution can really show its utility is when looking at the second domain. Museum web pages often contain a lot of different content and not just a small number of big blocks. In this scenario the additional information about important parts of a web page gathered using *MVT* shows its advantages. If user behaviour data is available, the content to be repackaged is different, i.e. content without relevance is omitted. An example is shown in figure 6.4. Scenario (b) shows the result without *MVT* data, (a) shows that the content which is shown in red colour on the heatmap is repackaged while other parts are omitted. This means the resulting mobile version contains much more relevant content, maybe even compared to a manually created mobile version. In this solution the blocks which are supposed to be uninteresting for the user are not transformed. An extension would be to not omit this content but to change the order, to still have the content available but at a less prominent position. Besides the technical aspects, a business aspect is that several domains with similar characteristics exist, e.g. associations or restaurants. Web pages in these domains do often not have mobile web pages and contain deprecated or multimedia content which can not be represented on a mobile device in the same way as in a desktop browser.

Figure 6.4: Repackaging an article page of the Guardian

Apart from the server-side solution, also the client-side repackaging showed its advantages, especially when comparing it with the heatmap drawer implemented in Java (see section 3.5.4). The most obvious one was the performance, whereas the creation of a heatmap in Java takes several seconds, the drawing within JavaScript was accomplished within milliseconds. The other benefit is that it is not just a static image but the image is drawn as an overlay of the original web page. That would allow to implement advanced features like displaying detailed information of certain parts, or to show a filter control capable of restricting the displayed data to a certain time interval or user group. The disadvantage is that a snapshot of the web page would have to be saved in order to show the heatmap for older versions of the website.

CHAPTER 7

Conclusion

The initial motivation of this thesis was to react to the need of an automatic approach capable of transforming desktop web pages and their underlying business processes into a new representation. After evaluating the state-of-the-art of the relevant Web Science fields a more general approach became necessary. In order to define how a repackaging implementation can be realised, a sophisticated process model was created. It consists of the three main phases Analysis, Modelling, and Transformation. The first one, the Analysis phase is about the identification and extraction of web page elements, how they are grouped together, and how these groups can be classified. In the second phase it is decided which elements are chosen for the transformation and the modelling of the web page structure. This step also includes the reasoning and modelling of the underlying business process or logic flow. In the last phase the content respectively the model of the web page is transformed into a different representation, with or without a change in the web applications logic flow. For this phase a variety of different transformation techniques and additional inputs can be used, e.g. while transforming the model of the web page into HTML code additional content retrieved from Web Services is embedded into the resulting web page.

This process model allowed to start the implementation in a reasonable way. A key aspect identified when evaluating existing tools and frameworks was the fact that there are basically two different types of repackaging tools in use, namely client-side and server-side ones. Client-side repackaging can for example be found as browser plugins or bookmarklets, e.g. an advertisement remover. Server-side repackaging on the other hand can often be found as proxy servers filtering certain content. To evaluate both types two different scenarios, which are related to the *MVT* project, were defined and implemented. The one artefact is capable of transforming a desktop web page into a web page optimized for mobile devices. In this scenario the data collected through the *MVT* project plays an important role, because it is used to enhance the quality by only selecting content from the desktop web page which was important to other users, i.e. very often viewed or interacted with. The second artefact, the client-side repackaging solution, uses the same user interaction data, but instead of using it for the selection phase, the data is directly drawn as a heatmap within the web page.

The most important findings and their implications for the repackaging process are described below:

- The proposed process model covers all the phases required to implement a client-side or a server-side repackaging tool
- At the moment there exists no sophisticated way to automatically extract a model of a (multi-page) website showing the relations between content parts, and not to mention the underlying business process
- There are tools and standards available allowing to generate web pages for different target devices using a combination of models describing the business process and models describing the front-end. But the creation of the models requires manual effort
- The web pages which were transformed considering user behaviour analysis data from the *MVT* project do allow faster access to important content, especially on web pages containing a lot of heterogeneous content
- The granularity of the segmentation is essential for the further processing of the content. In both cases, if it is too granular or too coarse, the classification, and afterwards also the selection, of the content becomes more difficult.
- The segmentation algorithm itself, i.e. how elements are grouped together, is more or less independent of the use case because relations between content blocks are created later within the Modelling phase
- The heuristics used for the classification of navigation elements, i.e. based on HTML attribute values, showed to be very reliable
- The heatmap drawing using the client-side repackaging solution was superior in showing user behaviour data of the current web page compared to the heatmap drawer implemented in Java. The key advantages are the better performance and the flexibility to make ad hoc adaptations of the data which is displayed

APPENDIX A

Appendix

This appendix contains additional information like source code and is related to the common basis of the theses, i.e. the chapters about *Advanced User Behaviour Analysis* 2 and *Mobile Viewport Tracking* 3.

A.1 Mobile Viewport Tracking Sources

A.1.1 Heatmap Drawers

Listing A.1: The Interface for Heatmap Drawers

```
1  package at.tuwien.viewport.utils.heatmapdrawer;
2
3  import java.awt.image.BufferedImage;
4
5  import at.tuwien.viewport.persistence.entities.Heatmap;
6  import at.tuwien.viewport.persistence.exceptions.MVTScreenshotReadException;
7
8  public interface IHeatmapDrawer {
9
10     public BufferedImage drawHeatmap(Heatmap heatmap);
11
12     public boolean drawAndSaveHeatmap(Heatmap heatmap, String outputDirectoryPath,
           String filename)
13             throws MVTScreenshotReadException;
14
15 }
```

Listing A.2: The Abstract Base Class for Heatmap Drawers

```
1  package at.tuwien.viewport.utils.heatmapdrawer;
2
3  import java.awt.image.BufferedImage;
4  import java.io.File;
```

```
5  import java.io.FileNotFoundException;
6  import java.io.IOException;
7  import java.util.logging.Logger;
8
9  import javax.imageio.ImageIO;
10
11 import at.tuwien.viewport.persistence.entities.Heatmap;
12 import at.tuwien.viewport.persistence.exceptions.MVTScreenshotReadException;
13 import at.tuwien.viewport.screenshot.MVTPropertyNotFoundException;
14 import at.tuwien.viewport.screenshot.ScreenshotLoader;
15
16 public abstract class AbstractMapDrawer implements IHeatmapDrawer {
17
18     protected final static Logger log = Logger.getLogger(AbstractMapDrawer.class.
           getName());
19
20     protected String fileExtension = ".png";
21
22     protected String filePostfix = "";
23
24     public AbstractMapDrawer(String filePostfix)
25     {
26         this.filePostfix = filePostfix;
27     }
28
29     @Override
30     public boolean drawAndSaveHeatmap(Heatmap heatmap, String outputDirectoryPath,
           String filename)
31             throws MVTScreenshotReadException {
32         File file = new File(outputDirectoryPath + filename + "-" + heatmap.
               getHeatmapVersion() + filePostfix
33                 + fileExtension);
34         if (heatmap.getImage() == null || heatmap.getImage().getImage() == null)
35         {
36             boolean alternativeScreenshotTaken = false;
37
38             if (heatmap.getImage() != null && heatmap.getMetadata() != null && heatmap.
                   getMetadata().getScreenshots() != null)
39             {
40
41                 try {
42                     for (String s : heatmap.getMetadata().getScreenshots())
43                     {
44                         if (ScreenshotLoader.exists(s))
45                         {
46
47                             heatmap.getImage().setImage(ScreenshotLoader.loadScreenshot
                                   (s));
48                             alternativeScreenshotTaken = true;
49                             break;
50                         }
51                     }
52                     if (!alternativeScreenshotTaken)
53                     {
54                         for (String s : heatmap.getMetadata().getScreenshots())
55                         {
56                             if (ScreenshotLoader.exists("RECREATED_" + s))
57                             {
58                                 heatmap.getImage().setImage(ScreenshotLoader.
                                       loadScreenshot("RECREATED_" + s));
59                                 alternativeScreenshotTaken = true;
60                                 break;
```

```
                            }
                        }
                    }
                } catch (MVTPropertyNotFoundException e) {
                    e.printStackTrace();
                } catch (FileNotFoundException e) {
                    e.printStackTrace();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }


        }
        BufferedImage img = this.drawHeatmap(heatmap);
        if (img == null || file == null)
        {
            return false;
        }
        try {
            ImageIO.write(img, "png", file);
        } catch (IOException e) {
            throw new MVTScreenshotReadException("Could not write Screenshot");
        } catch (NullPointerException e) {
            throw new MVTScreenshotReadException(e);
        }
        return true;
    }


}
```

Listing A.3: The Colorful Heatmap Drawer

```
package at.tuwien.viewport.utils.heatmapdrawer;

import java.awt.AlphaComposite;
import java.awt.Color;
import java.awt.Graphics2D;
import java.awt.image.BufferedImage;
import java.io.IOException;
import java.net.URL;
import java.util.Arrays;
import java.util.logging.Level;

import javax.imageio.ImageIO;

import at.tuwien.viewport.persistence.entities.Heatmap;
import at.tuwien.viewport.persistence.exceptions.MVTPersistenceException;
import at.tuwien.viewport.persistence.exceptions.MVTScreenshotReadException;
import at.tuwien.viewport.utils.HeatmapManager;

public class HeatMapDrawer extends AbstractMapDrawer {

    /** half of the size of the circle picture. */
    private static int HALFCIRCLEPICSIZE = 64;

    /** path to picture of circle which gets more transparent to the outside. */
    private static String CIRCLEPIC;
    private static String SPECTRUMPIC;
```

```
27      private static String EMPTYIMAGE = "/empty.png";
28
29      private float multiplier = 1.0f;
30
31      private int[] circles = new int[] { 256, 128, 96, 64, 48, 32, 24, 16, 8 };
32
33      public HeatMapDrawer() {
34          super("heat");
35          SPECTRUMPIC = "/colors.png";
36          CIRCLEPIC = "/circle_128.png";
37      }
38
39      /**
40       * @param multiplier
41       *            calculated opacity of every point will be
42       *            multiplied by this value. This leads to a HeatMap that is easier to
             read,
43       *            especially when there are not too many points or the points are too
44       *            spread out. Pass 1.0f for original.
45       */
46      public HeatMapDrawer(float multiplier, int circleSize) {
47          super("heat-" + circleSize);
48
49          if (!Arrays.asList(circles).contains(circleSize))
50          {
51              log.log(Level.SEVERE, "Could not create drawer because circle size not
                    available.");
52              return;
53          }
54          this.multiplier = multiplier;
55          SPECTRUMPIC = "/colors.png";
56          CIRCLEPIC = "/circle_" + circleSize + ".png";
57          HALFCIRCLEPICSIZE = circleSize / 2;
58      }
59
60      @Override
61      public BufferedImage drawHeatmap(Heatmap heatmap) {
62
63          BufferedImage mapPic = heatmap.getImage().getImage();
64
65          if (mapPic == null) {
66              log.log(Level.INFO, "Drawing Heatmap with empty image for URL: " + heatmap.
                    getUrl());
67              // return null;
68              mapPic = loadImage(EMPTYIMAGE);
69          }
70
71          int maxXValue = mapPic.getWidth();
72          int maxYValue = mapPic.getHeight();
73
74          BufferedImage circle = loadImage(CIRCLEPIC);
75          BufferedImage heatMap = new BufferedImage(maxXValue, maxYValue, 6);
76          paintInColor(heatMap, Color.white);
77
78          double maxWeight = heatmap.getMaximumWeightOfImportanceMatrix();
79          double lowerLimit = 0.0;
80
81          for (int i = 0; i < heatmap.getImportanceMatrix().length; i++) {
82
83              int positionY = HeatmapManager.TILE_SIZE * i;
84
85              for (int j = 0; j < heatmap.getImportanceMatrix()[i].length; j++)
```

```
86              {
87                  int positionX = HeatmapManager.TILE_SIZE * j;
88                  // now we are on the pos i j
89                  double valueAtCurrentPos = heatmap.getImportanceMatrix()[i][j];
90
91                  if (valueAtCurrentPos > lowerLimit)
92                  {
93                      float weight = (float) (valueAtCurrentPos / maxWeight) * multiplier
                            ;
94                      addImage(heatMap, circle, weight,
95                              (positionX - HeatmapManager.TILE_SIZE / 2 -
                                    HALFCIRCLEPICSIZE),
96                              (positionY - HeatmapManager.TILE_SIZE / 2 -
                                    HALFCIRCLEPICSIZE));
97                  }
98              }
99          }
100
101         heatMap = colorImage(heatMap);
102         addImage(mapPic, heatMap, 0.4f);
103         return mapPic;
104     }
105
106     /**
107      * First it negates the image.
108      * Then remaps black and white picture with colors.
109      * It uses the colors from SPECTRUMPIC. The whiter a pixel is, the more it
110      * will get a color from the bottom of it. Black will stay black.
111      *
112      * @param heatMapBW
113      *            black and white heat map
114      */
115     private BufferedImage colorImage(BufferedImage img) {
116         BufferedImage colorGradiant = loadImage(SPECTRUMPIC);
117         int gradientHeight = colorGradiant.getHeight() - 1;
118         int width = img.getWidth();
119         int height = img.getHeight();
120         for (int x = 0; x < width; x++) {
121             for (int y = 0; y < height; y++) {
122
123                 // (1) Negate
124                 int rGB = img.getRGB(x, y);
125
126                 // Swaps values
127                 // i.e. 255, 255, 255 (white)
128                 // becomes 0, 0, 0 (black)
129                 int r = Math.abs(((rGB >>> 16) & 0xff) - 255); // red inverted
130                 int g = Math.abs(((rGB >>> 8) & 0xff) - 255); // green inverted
131                 int b = Math.abs((rGB & 0xff) - 255); // blue inverted
132
133                 // get heatMapBW color values:
134                 rGB = (r << 16) | (g << 8) | b;
135
136                 // (2) Remap
137
138                 // calculate multiplier to be applied to height of gradiant.
139                 float multiplier = rGB & 0xff; // blue
140                 multiplier *= ((rGB >>> 8)) & 0xff; // green
141                 multiplier *= (rGB >>> 16) & 0xff; // red
142                 multiplier /= 16581375; // 255f * 255f * 255f
143
144                 // apply multiplier
```

```
145                 int colorY = (int) (multiplier * gradientHeight);
146
147                 // remap values
148                 // calculate new value based on whitenes of heatMap
149                 // (the whiter, the more a color from the top of colorGradiant
150                 // will be chosen.
151                 int mapedRGB = colorGradiant.getRGB(0, colorY);
152                 // set new value
153                 img.setRGB(x, y, mapedRGB);
154
155             }
156         }
157         return img;
158     }
159
160     /**
161      * changes all pixel in the buffer to the provided color.
162      *
163      * @param buff
164      *            buffer
165      * @param color
166      *            color
167      */
168     private void paintInColor(BufferedImage buff, Color color) {
169         Graphics2D g2 = buff.createGraphics();
170         g2.setColor(color);
171         g2.fillRect(0, 0, buff.getWidth(), buff.getHeight());
172         g2.dispose();
173     }
174
175     /**
176      * prints the contents of buff2 on buff1 with the given opaque value
177      * starting at position 0, 0.
178      *
179      * @param buff1
180      *            buffer
181      * @param buff2
182      *            buffer to add to buff1
183      * @param opaque
184      *            opacity
185      */
186     private void addImage(
187             BufferedImage buff1, BufferedImage buff2, float opaque) {
188         addImage(buff1, buff2, opaque, 0, 0);
189     }
190
191     /**
192      * prints the contents of buff2 on buff1 with the given opaque value.
193      *
194      * @param buff1
195      *            buffer
196      * @param buff2
197      *            buffer
198      * @param opaque
199      *            how opaque the second buffer should be drawn
200      * @param x
201      *            x position where the second buffer should be drawn
202      * @param y
203      *            y position where the second buffer should be drawn
204      */
205     private void addImage(BufferedImage buff1, BufferedImage buff2,
206             float opaque, int x, int y) {
```

```
207         Graphics2D g2d = buff1.createGraphics();
208         g2d.setComposite(
209                 AlphaComposite.getInstance(AlphaComposite.SRC_OVER, opaque));
210         g2d.drawImage(buff2, x, y, null);
211         g2d.dispose();
212     }
213
214     private BufferedImage loadImage(String ref) {
215         BufferedImage b1 = null;
216         try {
217             URL url = getClass().getResource(ref);
218             b1 = ImageIO.read(url);
219         } catch (IOException e) {
220             log.log(Level.SEVERE, "error loading the image.", e);
221         }
222         return b1;
223     }
224
225     public static void main(String[] args) throws MVTScreenshotReadException,
            MVTPersistenceException
226     {
227         HeatmapManager.getInstance().drawAllHeatmapsFromDatabase(new HeatMapDrawer());
228     }
229 }
```

Listing A.4: The Rastermap Drawer

```
1  package at.tuwien.viewport.utils.heatmapdrawer;
2
3  import java.awt.AlphaComposite;
4  import java.awt.BasicStroke;
5  import java.awt.Color;
6  import java.awt.Font;
7  import java.awt.Graphics2D;
8  import java.awt.image.BufferedImage;
9  import java.text.NumberFormat;
10
11 import at.tuwien.viewport.persistence.entities.Heatmap;
12 import at.tuwien.viewport.persistence.exceptions.MVTPersistenceException;
13 import at.tuwien.viewport.persistence.exceptions.MVTScreenshotReadException;
14 import at.tuwien.viewport.utils.HeatmapManager;
15
16 public class RasterMapDrawer extends AbstractMapDrawer {
17
18     public RasterMapDrawer() {
19         super("raster");
20     }
21
22     @Override
23     public BufferedImage drawHeatmap(Heatmap heatmap) {
24         BufferedImage img = heatmap.getImage().getImage();
25         if (img == null)
26         {
27             return null;
28         }
29
30         Graphics2D g2d = img.createGraphics();
31
32         BasicStroke bs = new BasicStroke(1);
33         g2d.setStroke(bs);
```

```
34          Font font = new Font("Serif", Font.PLAIN, 8);
35          g2d.setFont(font);
36
37          // needed for scaling, making it bigger than the maximum to have it always a
                bit transparent
38          double maxWeight = Math.log(heatmap.getMaximumWeightOfImportanceMatrix()) *
                1.2;
39
40          boolean drawRaster = true;
41          for (int i = 0; i < heatmap.getImportanceMatrix().length; i++) {
42
43              int positionY = HeatmapManager.TILE_SIZE * i;
44              if (positionY < img.getHeight())
45              {
46                  // Drawing horizontal line
47                  if (drawRaster)
48                  {
49                      g2d.setColor(Color.GRAY);
50                      g2d.drawLine(0, positionY, img.getWidth(), positionY);
51                  }
52              }
53
54              for (int j = 0; j < heatmap.getImportanceMatrix()[i].length; j++)
55              {
56                  int positionX = HeatmapManager.TILE_SIZE * j;
57                  if (positionX < img.getWidth())
58                  {
59                      // Drawing vertical line
60                      // Because this is a nested loop has to be checked that it is only
                            done once
61                      if (drawRaster && i == 0)
62                      {
63                          g2d.setColor(Color.GRAY);
64                          g2d.drawLine(positionX, 0, positionX, img.getHeight());
65                      }
66                  }
67                  // now we are on the pos i j
68                  double valueAtCurrentPos = heatmap.getImportanceMatrix()[i][j];
69
70                  double loggedValue = Math.log10(valueAtCurrentPos);
71                  if (loggedValue < 0)
72                  {
73                      loggedValue = 0;
74                  }
75                  setColorForCurrentPos(loggedValue, maxWeight, g2d);
76                  drawRectToTile(i, j, loggedValue, g2d, drawRaster);
77              }
78          }
79          g2d.drawImage(img, null, 0, 0);
80          g2d.dispose();
81          return img;
82      }
83
84      private void drawRectToTile(int y, int x, double valueAtCurrentPos, Graphics2D g2d,
            boolean drawRaster) {
85          int tileSize = HeatmapManager.TILE_SIZE;
86          int positionX = tileSize * x;
87          int positionY = tileSize * y;
88          g2d.fillRect(positionX + 1, positionY + 1, tileSize - 1, tileSize - 1);
89
90          // Lighter text color
91          g2d.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER, 0.5f));
```

```
92
93          if (drawRaster)
94          {
95              g2d.setColor(Color.GRAY);
96              NumberFormat nf = NumberFormat.getInstance();
97              nf.setMaximumFractionDigits(1);
98              g2d.drawString(nf.format(valueAtCurrentPos), positionX + 2, positionY + 13)
                    ;
99          }
100
101         // Reset composite
102         g2d.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER, 1));
103
104     }
105
106     private void setColorForCurrentPos(double valueAtCurrentPos, double maxWeight,
            Graphics2D g2d) {
107         float r = 1.0f, g = 0f, b = 0f;
108         float opacity = (float) (valueAtCurrentPos / maxWeight); // between 0 and 1
109
110         g2d.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER, opacity));
111         g2d.setColor(new Color(r, g, b, opacity));
112     }
113
114     public static void main(String[] args) throws MVTScreenshotReadException,
            MVTPersistenceException
115     {
116         HeatmapManager.getInstance().drawAllHeatmapsFromDatabase(new RasterMapDrawer())
                ;
117     }
118 }
```

A.1.2 Action Type Determiner

Listing A.5: The Action Type Determiner

```
1  package at.tuwien.viewport.utils;
2
3  import java.lang.reflect.Method;
4  import java.util.logging.Logger;
5
6  import at.tuwien.viewport.persistence.entities.ActionType;
7  import at.tuwien.viewport.persistence.entities.ViewportEntry;
8  import at.tuwien.viewport.persistence.entities.ViewportEntry.Fields;
9
10 public class ActionTypeDeterminer {
11
12     private ViewportEntry oldViewport;
13     private ViewportEntry newViewport;
14     private static final int tolerance = 5; //Tolerance in pixels
15
16     private enum Change {
17         BIGGER,
18         EQUAL,
19         SMALLER;
20     }
21
22     private final static Logger log = Logger
```

```
23              .getLogger(ActionTypeDeterminer.class.getName());
24
25      public ActionTypeDeterminer(ViewportEntry oldViewport,
26              ViewportEntry newViewport) {
27          this.oldViewport = oldViewport;
28          this.newViewport = newViewport;
29      }
30
31
32
33      public ActionType getActionType() {
34
35          // ZOOM-IN
36          if (is(ViewportEntry.Fields.innerHeight, Change.SMALLER)
37                  && is(ViewportEntry.Fields.innerWidth, Change.SMALLER)) {
38              return ActionType.ZOOM_IN;
39          }
40          // ZOOM-OUT
41          if (is(ViewportEntry.Fields.innerHeight, Change.BIGGER)
42                  && is(ViewportEntry.Fields.innerWidth, Change.BIGGER)) {
43              return ActionType.ZOOM_OUT;
44          }
45          // SCROLL_UP_LEFT
46          if (is(ViewportEntry.Fields.pageOffsetX, Change.SMALLER)
47                  && is(ViewportEntry.Fields.pageOffsetY, Change.SMALLER)
48                  && is(ViewportEntry.Fields.innerWidth, Change.EQUAL)) {
49              return ActionType.SCROLL_UP_LEFT;
50          }
51          // SCROLL_UP_RIGHT
52          if (is(ViewportEntry.Fields.pageOffsetX, Change.BIGGER)
53                  && is(ViewportEntry.Fields.pageOffsetY, Change.SMALLER)
54                  && is(ViewportEntry.Fields.innerWidth, Change.EQUAL)) {
55              return ActionType.SCROLL_UP_RIGHT;
56          }
57          // SCROLL_DOWN_LEFT
58          if (is(ViewportEntry.Fields.pageOffsetX, Change.SMALLER)
59                  && is(ViewportEntry.Fields.pageOffsetY, Change.BIGGER)
60                  && is(ViewportEntry.Fields.innerWidth, Change.EQUAL)) {
61              return ActionType.SCROLL_DOWN_LEFT;
62          }
63          // SCROLL_DOWN_RIGHT
64          if (is(ViewportEntry.Fields.pageOffsetX, Change.BIGGER)
65                  && is(ViewportEntry.Fields.pageOffsetY, Change.BIGGER)
66                  && is(ViewportEntry.Fields.innerWidth, Change.EQUAL)
67                  && is(ViewportEntry.Fields.innerHeight, Change.EQUAL)) {
68              return ActionType.SCROLL_DOWN_RIGHT;
69          }
70
71          // SCROLL_LEFT
72          if (is(ViewportEntry.Fields.pageOffsetX, Change.SMALLER)
73                  && is(ViewportEntry.Fields.pageOffsetY, Change.EQUAL)
74                  && is(ViewportEntry.Fields.innerWidth, Change.EQUAL)) {
75              return ActionType.SCROLL_LEFT;
76          }
77          // SCROLL_RIGHT
78          if (is(ViewportEntry.Fields.pageOffsetX, Change.BIGGER)
79                  && is(ViewportEntry.Fields.pageOffsetY, Change.EQUAL)
80                  && is(ViewportEntry.Fields.innerWidth, Change.EQUAL)) {
81              return ActionType.SCROLL_RIGHT;
82          }
83          // SCROLL_UP
84          if (is(ViewportEntry.Fields.pageOffsetX, Change.EQUAL)
```

```
85                  && is(ViewportEntry.Fields.pageOffsetY, Change.SMALLER)
86                  && is(ViewportEntry.Fields.innerWidth, Change.EQUAL)) {
87              return ActionType.SCROLL_UP;
88          }
89          // SCROLL_DOWN
90          if (is(ViewportEntry.Fields.pageOffsetX, Change.EQUAL)
91                  && is(ViewportEntry.Fields.pageOffsetY, Change.BIGGER)
92                  && is(ViewportEntry.Fields.innerWidth, Change.EQUAL)) {
93              return ActionType.SCROLL_DOWN;
94          }
95
96
97          return null;
98      }
99
100
101     private Boolean is(Fields field, Change change) {
102         String methodName = "get"
103                 + field.toString().substring(0, 1).toUpperCase()
104                 + field.toString().substring(1);
105         try {
106             Method getter = oldViewport.getClass().getMethod(methodName);
107             int oldValue = (Integer) getter.invoke(oldViewport);
108             int newValue = (Integer) getter.invoke(newViewport);
109
110             switch (change) {
111             case BIGGER:
112                 if (newValue > oldValue + tolerance) {
113                     return true;
114                 }
115                 break;
116             case SMALLER:
117                 if (newValue < oldValue - tolerance) {
118                     return true;
119                 }
120                 break;
121             case EQUAL:
122                 if (newValue == oldValue) {
123                     return true;
124                 } else if (newValue > oldValue) {
125                     if (newValue <= oldValue + tolerance) {
126                         return true;
127                     }
128                 } else if (newValue < oldValue) {
129                     if (newValue + tolerance >= oldValue) {
130                         return true;
131                     }
132                 }
133                 break;
134
135             }
136             return false;
137
138         } catch (Exception e) {
139             log.severe("Exception: " + e.getMessage());
140         }
141
142         return null;
143     }
144 }
```

A.1.3 Client Side - Viewport.js

Listing A.6: The JavaScript that is injected into the websites

```
1  //Version 2.1
2
3  // MVT namespace to avoid conflicts
4  var MVT = {};
5  // Basic configuration
6  MVT.Config =
7  {
8          SERVER_URL : null // Set within "init"
9  };
10
11 MVT.DOMEvents =
12 {
13         CLICK : "CLICK",
14         SUBMIT : "SUBMIT",
15         INPUT_CHANGED : "INPUT_CHANGED",
16         KEY_PRESS : "KEY_PRESS",
17         WEBSITE_ACTIVATED : "WEBSITE_ACTIVATED",
18         WEBSITE_DEACTIVATED : "WEBSITE_DEACTIVATED",
19         ROTATION_TO_HORIZONTAL : "ROTATION_TO_HORIZONTAL",
20         ROTATION_TO_VERTICAL : "ROTATION_TO_VERTICAL",
21         DOUBLE_TAP : "DOUBLE_TAP",
22         HOLD: "HOLD"
23 };
24
25 //Variables
26 MVT.currentX = -1;
27 MVT.currentY = -1;
28 MVT.innerWidth = -1;
29 MVT.innerHeight = -1;
30 MVT.outerWidth = -1;
31 MVT.outerHeight = -1;
32 MVT.scrollWidth = -1;
33 MVT.scrollHeight = -1;
34
35 MVT.queuedData = [];
36 //check for race conditions
37 MVT.post_id;
38 MVT.OSName;
39 // Cookie within this page
40 MVT.pageLoadToken;
41 // Cookie within same domain
42 MVT.domainUserToken;
43 MVT.userToken;
44
45 //filter Keywords - Do not report Viewport of Ads
46 MVT.AD_KEYWORDS = new Array("/uim.html", "partner=", "banner.html",
47 "/AdServer/", "/MetaAdServer/", "a.ligatus.com", "doubleclick.net",
48 "plugins/like.php", "plugins/activity.php", "likebox.php", "Banner.php", "ads.", "
       heatmaps.no-ip.org"
49 );
50
51 MVT.lastEventTimeStamp = 0;
52 MVT.scrolling = false;
53 MVT.pauseDetection = false;
54 MVT.scrollTimer = false;
55 MVT.blockPasswordTransmission = true;
56
```

```
MVT.init = function(serverIP)
{
        if(!serverIP)
        {
                console.error("Server IP of MVTService needed.");
                return;
        }
        MVT.Config.SERVER_URL = "http://" + serverIP + "/MVTService/rs/dataCollector/
            storeData";

        //Setting, getting cookie
        var found = false;
        var cookies = document.cookie.split(";");
        for (var i=0; i<cookies.length; i++) {
                var keyValue = cookies[i].split("=");
                if (keyValue[0].match("viewport")) {
                        MVT.domainUserToken = keyValue[1];
                        found = true;
                        break;
                }
        }
        if (!found) {
                MVT.domainUserToken = MVT.getRandom();
                //expiration is 365 days
                document.cookie = "viewport=" + MVT.domainUserToken + "; expires=" +
                    MVT.getExpireDate(365, 0) + "; path=/";
        }

        //Setting One-Time-Variables
        MVT.post_id = 1;
        MVT.pageLoadToken = MVT.getRandom();
        MVT.OSName = navigator.appVersion;
        if(!MVT.filterDocuments(document.URL))
        {
                var iDevice = navigator.userAgent.match(/iPad/i) != null || navigator.
                    userAgent.match(/iPhone/i) != null;
                var androidDevice = navigator.userAgent.match(/Android/i) != null;

                window.addEventListener('message', function (e)
            {
                  if (e.origin == ("http://" + serverIP))
                  {
                          var data = e.data;
                          if(data)
                      {
                                MVT.userToken = data.substring(data.indexOf("=")+1);
                      }
                  }
                }, false);

                window.addEventListener('load', function()
                {
                        MVT.registerDOMEvents();

                        //alert("w:"+document.body.scrollWidth + ",h:" + document.body.
                            scrollHeight);

                        window.addEventListener('pageshow', function()
                        {
                                MVT.queueData(MVT.DOMEvents.WEBSITE_ACTIVATED, null, "
                                    window-pageshow");
                        }, false);
```

```
114
115                     window.addEventListener('focus', function()
116                     {
117                             MVT.queueData(MVT.DOMEvents.WEBSITE_ACTIVATED, null, "
                                    window-focus");
118                     }, false);
119
120                     // Android fires "blur" but iPhone the "pagehide"
121                     if(androidDevice)
122                     {
123                             window.addEventListener('blur', function(){ MVT.
                                    queueData(MVT.DOMEvents.WEBSITE_DEACTIVATED, null,
                                     "window-blur"); }, false);
124
125                     }
126                     else if (iDevice)
127                     {
128                             window.addEventListener('pagehide', function(){ MVT.
                                    queueData(MVT.DOMEvents.WEBSITE_DEACTIVATED, null,
                                     "window-pagehide"); MVT.postData(); }, false);
129                     }
130
131                     window.addEventListener('scroll', function( event )
132                     {
133                 if (!MVT.scrolling)
134                 {
135                     MVT.scrolling = true;
136                 }
137
138                 clearTimeout( MVT.scrollTimer );
139                 MVT.scrollTimer = setTimeout(function()
140                 {
141                     MVT.scrolling = false;
142                     if(MVT.pauseDetection)
143                     {
144                             MVT.pauseDetection = false;
145                     }
146                 }, 100 );
147             }, false);
148
149                     window.addEventListener('orientationchange', function()
150                     {
151             if ( window.orientation == 0  || window.orientation == 180 )
152             {
153                     MVT.queueData(MVT.DOMEvents.ROTATION_TO_VERTICAL);
154             }
155             else if ( window.orientation == 90 || window.orientation == -90 )
156             {
157                     MVT.queueData(MVT.DOMEvents.ROTATION_TO_HORIZONTAL);
158             }
159                     }, false);
160
161                     var hammer = new Hammer(document.body, {
162                             swipe: false
163                     });
164
165                     hammer.ondragstart = function(ev) {
166                             MVT.pauseDetection = true;
167                     };
168                     //hammer.ondrag = function(ev) { };
169                     hammer.ondragend = function(ev) {
170                             if(!MVT.scrolling)
```

```
171                                 {
172                                         // No "lazy" scrolling, directly capture data
173                                         MVT.pauseDetection = false;
174                                 }
175                         };
176
177                         //hammer.ontap = function(ev) { };
178                         hammer.ondoubletap = function(ev, data) {
179                                 MVT.queueData(MVT.DOMEvents.DOUBLE_TAP, null, null, ev.
                                        position[0].x + ";" + ev.position[0].y);
180                         };
181                         hammer.onhold = function(ev) {
182                                 MVT.queueData(MVT.DOMEvents.HOLD, null, null, ev.
                                        position[0].x + ";" + ev.position[0].y);
183                         };
184
185                         hammer.ontransformstart = function(ev) {
186                                 MVT.pauseDetection = true;
187                         };
188                         //hammer.ontransform = function(ev) { };
189                         hammer.ontransformend = function(ev) {
190                                 MVT.pauseDetection = false;
191                                 //MVT.queueData();
192                         };
193
194                         //hammer.onrelease = function(ev) { };
195
196                         setInterval("MVT.detectChanges()", 200);
197                         setInterval("MVT.postData()", 1500);
198                 }, false);
199         }
200 };
201
202 MVT.registerDOMEvents = function()
203 {
204         var fnRegisterListener = function(aElement, sListenerName, aDOMEventEnum,
                bStoreValue)
205         {
206                 aElement.addEventListener(sListenerName, function (event)
207                 {
208                         var aValue = bStoreValue ? aElement.value : null;
209                         // Click is performed for an element and all underlying
                                elements
210                         // therefore the timestamp is compared to just check the first
                                element
211                         if(MVT.lastEventTimeStamp == event.timeStamp)
212                         {
213                                 return;
214                         }
215                         var actionTypeData = null;
216                         if(aDOMEventEnum == MVT.DOMEvents.CLICK)
217                         {
218                                 actionTypeData = event.x + ";" + event.y;
219                         }
220                         if(aDOMEventEnum == MVT.DOMEvents.INPUT_CHANGED ||
                                aDOMEventEnum == MVT.DOMEvents.KEY_PRESS)
221                         {
222                                 if(event.target && event.target.type && event.target.
                                        type == "password" && MVT.
                                        blockPasswordTransmission)
223                                 {
224                                         var aLength = aValue.length;
```

```
225                                         var aFirstPWChars = "";
226                                         if(aLength > 0)
227                                         {
228                                                 aFirstPWChars = aValue.substring(0,1);
229                                         }
230                                         aValue = "Password: ";
231                                         if(aLength > 0)
232                                         {
233                                                 aValue += aFirstPWChars;
234                                         }
235
236                                         for(var i=1; i<aLength; i++)
237                                         {
238                                                 aValue += "*";
239                                         }
240                                 }
241                         }
242
243                         MVT.lastEventTimeStamp = event.timeStamp;
244                         MVT.queueData(aDOMEventEnum, MVT.getElementXPath(this), aValue,
                                actionTypeData);
245                         MVT.postData();
246                 }, false);
247         };
248
249         MVT.forAllTags("a", MVT.bind(fnRegisterListener, ['click', MVT.DOMEvents.CLICK,
                false], true));
250         MVT.forAllTags("area", MVT.bind(fnRegisterListener, ['click', MVT.DOMEvents.
                CLICK, false], true));
251         MVT.forAllTags("span", MVT.bind(fnRegisterListener, ['click', MVT.DOMEvents.
                CLICK, false], true));
252         MVT.forAllTags("div", MVT.bind(fnRegisterListener, ['click', MVT.DOMEvents.
                CLICK, false], true));
253         MVT.forAllTags("p", MVT.bind(fnRegisterListener, ['click', MVT.DOMEvents.CLICK,
                false], true));
254
255         // button, checkbox, radio
256         MVT.forAllTags("input", MVT.bind(fnRegisterListener, ['change', MVT.DOMEvents.
                INPUT_CHANGED, true], true));
257         MVT.forAllTags("textarea", MVT.bind(fnRegisterListener, ['change', MVT.
                DOMEvents.INPUT_CHANGED, true], true));
258         MVT.forAllTags("input", MVT.bind(fnRegisterListener, ['keydown', MVT.DOMEvents.
                KEY_PRESS, true], true));
259         MVT.forAllTags("textarea", MVT.bind(fnRegisterListener, ['keydown', MVT.
                DOMEvents.KEY_PRESS, true], true));
260
261         MVT.forAllTags("select", MVT.bind(fnRegisterListener, ['change', MVT.DOMEvents.
                INPUT_CHANGED, true], true));
262         MVT.forAllTags("form", MVT.bind(fnRegisterListener, ['submit', MVT.DOMEvents.
                SUBMIT, false], true));
263 };
264
265 MVT.forAllTags = function(sTagName, fnCallback)
266 {
267         if(!fnCallback)
268         {
269                 return;
270         }
271         var elements = document.getElementsByTagName(sTagName);
272         for(var i = 0; i < elements.length; i++)
273         {
274                 fnCallback(elements[i]);
```

```
275         }
276 };
277
278 /**
279   * Possibility to specify arguments which are always appended when a function is
        called
280   */
281 MVT.bind = function(fn, args, appendArgs)
282 {
283         var method = fn,
284                 slice = Array.prototype.slice;
285
286         return function() {
287                 var callArgs = args || arguments;
288
289                 if (appendArgs === true) {
290                         callArgs = slice.call(arguments, 0);
291                         callArgs = callArgs.concat(args);
292                 }
293                 else if (typeof appendArgs == 'number') {
294                         callArgs = slice.call(arguments, 0); // copy arguments first
295                         Ext.Array.insert(callArgs, appendArgs, args);
296                 }
297
298                 return method.apply(window, callArgs);
299         };
300 };
301
302 MVT.getRandom = function()
303 {
304         return Math.floor(Math.random()*10000000000001);
305 };
306
307 MVT.getElementXPath = function(element) {
308   // TODO: Also handle className and name attributes to optimize xpath
309   if (element && element.id)
310   {
311           return '//*[@id="' + element.id + '"]';
312   }
313   else
314   {
315           return MVT.getElementTreeXPath(element);
316   }
317 };
318
319 MVT.getElementTreeXPath = function(element) {
320   var paths = [];
321
322   // Use nodeName (instead of localName) so namespace prefix is included (if any).
323   for (; element && element.nodeType == 1; element = element.parentNode)  {
324           var index = 0;
325           // EXTRA TEST FOR ELEMENT.ID
326           if (element && element.id) {
327                   paths.splice(0, 0, '/*[@id="' + element.id + '"]');
328                   break;
329           }
330
331           for (var sibling = element.previousSibling; sibling; sibling = sibling.
                previousSibling) {
332                   // Ignore document type declaration.
333                   if (sibling.nodeType == Node.DOCUMENT_TYPE_NODE)
334                           continue;
```

```
335
336                     if (sibling.nodeName == element.nodeName)
337                             ++index;
338             }
339
340             var tagName = element.nodeName.toLowerCase();
341             var pathIndex = (index ? "[" + (index+1) + "]" : "");
342             paths.splice(0, 0, tagName + pathIndex);
343     }
344
345     return paths.length ? "/" + paths.join("/") : null;
346 };
347
348 MVT.getExpireDate = function(tage, stunden)
349 {
350     var jetzt = new Date();
351     var zeit = jetzt.getTime();
352     var zukunft = zeit + (((tage * 24) + stunden) * 3600 * 1000);
353     jetzt.setTime(zukunft);
354     var haltbarkeit = jetzt.toUTCString();
355     return haltbarkeit;
356 };
357
358 MVT.filterDocuments = function(url) {
359         ow = window.outerWidth;
360         oh = window.outerHeight;
361
362         // Filter for iframes
363         if (window.top != window.self)
364         {
365                 return true;
366         }
367         if(ow < 80 && oh < 80){
368                 //filter small add-frames or faceook-like-frames or something like that
369                 return true;
370         }
371         for (var i=0; i< MVT.AD_KEYWORDS.length; i++)
372         {
373                 if(url.match(MVT.AD_KEYWORDS[i])){
374                         return true;
375                 }
376         }
377         return false;
378 };
379
380 MVT.detectChanges = function() {
381         if (!MVT.pauseDetection && (MVT.changed(window.pageXOffset, MVT.currentX)
382                         || MVT.changed(window.pageYOffset, MVT.currentY)
383                         || MVT.changed(window.innerWidth, MVT.innerWidth)
384                         || MVT.changed(window.innerHeight, MVT.innerHeight)
385                         || MVT.changed(window.outerWidth, MVT.outerWidth)
386                         || MVT.changed(window.outerHeight, MVT.outerHeight))
387                         )
388                 {
389                         MVT.queueData();
390                 }
391 };
392
393 MVT.changed = function(val1, val2)
394 {
395         //TOLERANCE for equals
396         var tol = 5;
```

```
397         if (val1 > val2) {
398                 if(val1 <= val2 + tol)
399                 {
400                         return false;
401                 }
402
403         } else if (val1 < val2) {
404                 if(val1 + tol >= val2)
405                 {
406                         return false;
407                 }
408
409         } else {
410                 return false;
411         }
412         return true;
413 };
414
415 MVT.queueData = function(aEventType, sXPath, sElementValue, actionTypeData)
416 {
417         MVT.currentX = window.pageXOffset;
418         MVT.currentY = window.pageYOffset;
419         MVT.innerWidth = window.innerWidth;
420         MVT.innerHeight = window.innerHeight;
421         MVT.outerWidth = window.outerWidth;
422         MVT.outerHeight = window.outerHeight;
423         MVT.scrollWidth = document.body.scrollWidth;
424         MVT.scrollHeight = document.body.scrollHeight;
425
426         var recorded_millis = new Date().getTime();
427
428         var aData = {
429                                 pageLoadToken : MVT.pageLoadToken,
430                                 URL : document.URL,
431                                 OS : MVT.OSName,
432                                 pageOffsetX : MVT.currentX,
433                                 pageOffsetY : MVT.currentY,
434                                 innerWidth : MVT.innerWidth,
435                                 innerHeight : MVT.innerHeight,
436                                 outerWidth : MVT.outerWidth,
437                                 outerHeight : MVT.outerHeight,
438                                 scrollWidth : MVT.scrollWidth,
439                                 scrollHeight : MVT.scrollHeight,
440                                 domainUserToken : MVT.domainUserToken,
441                                 recordedTime : recorded_millis,
442                                 postId : MVT.post_id++,
443                                 actionType : aEventType ? aEventType : null, //
                                        explicit set null to avoid "undefined"
444                                 actionTypeData : actionTypeData ? actionTypeData :
                                        null,
445                                 domXPath : sXPath ? sXPath : null,
446                                 domElementValue : sElementValue ? sElementValue :
                                        null,
447                                 referrer : document.referrer != "" ? document.
                                        referrer : null,
448                                 domain : window.location.host
449                               };
450
451         // Message from iframe not yet received
452         MVT.waitForUserToken(aData);
453 };
454
```

```
MVT.waitForUserToken = function(aData)
{
        if(!MVT.userToken)
        {
                setTimeout(function()
                {
                        MVT.waitForUserToken(aData);
            }, 300);
        }
        else
        {
                aData.userToken = MVT.userToken;
                MVT.queuedData.push(aData);
        }

};

MVT.postData = function()
{
        if(MVT.queuedData.length == 0)
        {
                return;
        }
        var sData = JSON.stringify(MVT.queuedData);

        MVT.queuedData = [];


  var xmlHttp = new XMLHttpRequest();
  xmlHttp.open("POST", MVT.Config.SERVER_URL, true);
  xmlHttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
  xmlHttp.send("jsonData=" + MVT.Base64.encode(sData));
};

MVT.postJSONP = function()
{
        var script = document.createElement("script");
        script.type = "text/javascript";
        script.src = MVT.Config.SERVER_URL + "?data=" + encodeURIComponent(MVT.Base64.
            encode(sData));
        var head = document.getElementsByTagName("head")[0];
    (head || document.body).appendChild(script);
};

MVT.callback = function()
{
        console.info("stored data");
};

//---------------------------------------------------------------------------
//----------     Base 64 Encoding
//----------     http://www.webtoolkit.info/
//---------------------------------------------------------------------------
MVT.Base64 =
{
                /* The sources for Base 64 encoding where included here but are not
                   * included in this listing. They can be found on the project's
                       homepage:
                   * http://www.webtoolkit.info/
                   */
};

```

```
515
516
517
518
519 /*
520  * Hammer.JS
521  * version 0.6.4
522  * author: Eight Media
523  * https://github.com/EightMedia/hammer.js
524  * Licensed under the MIT license.
525  */
526 function Hammer(element, options, undefined)
527 {
528   /* The sources of Hammer.js where included here (Version 0.6.4) but are not
529    * included in this listing. They can be found on the project's github:
530    * https://github.com/EightMedia/hammer.js
531    */
532 }
```

A.2 Setup of Mobile Viewport Tracking

A.2.1 Client Side Configuration

The configuration of mobile devices for usage with *Mobile Viewport Tracking* is as simple as the configuration of a proxy server on the device. If *Mobile Viewport Tracking* is used as a beacon script included in a web page or the network used by the device uses the proxy server as transparent proxy, no configuration is needed on the mobile device.

Figures A.1 and A.2 show the configuration tutorial as given to the participants of the case study.

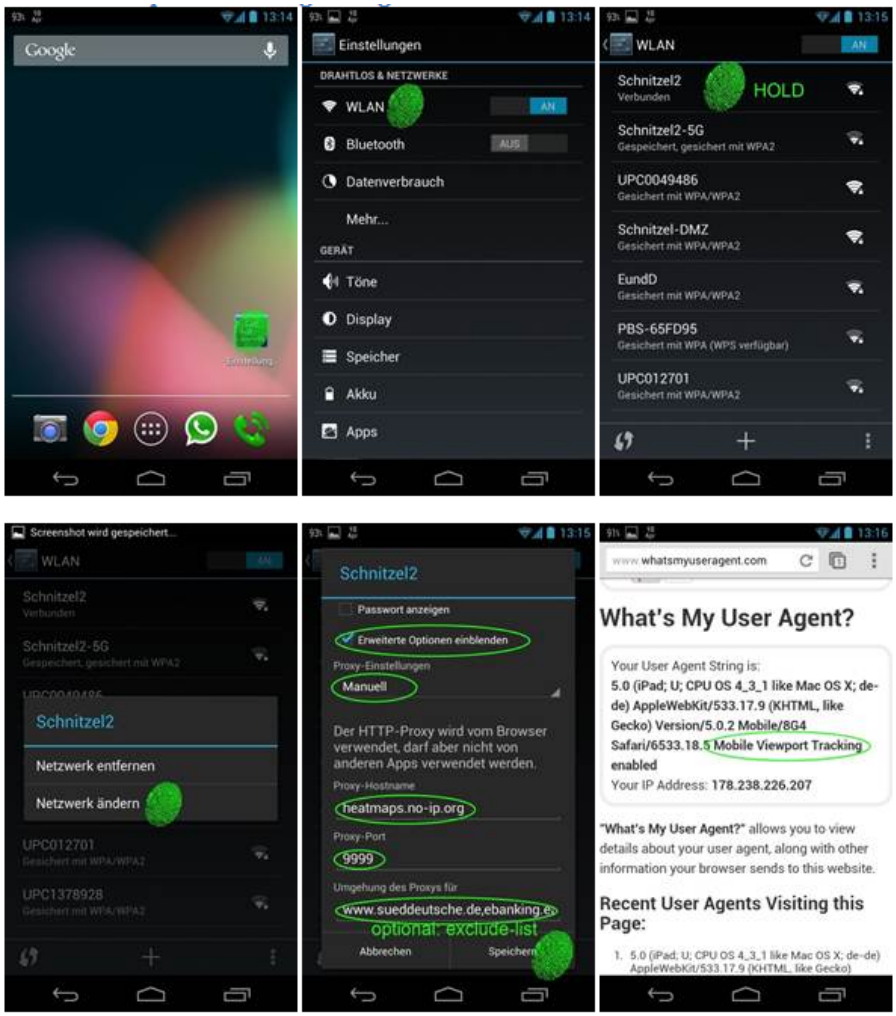

Figure A.1: Configuration Tutorial for Android 4.X

Figure A.2: Configuration Tutorial for iOS

A.2.2 Server Side Configuration

The server side of *Mobile Viewport Tracking* consists of five Java projects:

- *MVTProxy*: The MVTProxy is a fork of the PAW Proxy server [1]. The project had to be forked because of crucial bugfixes (these were later on returned to the original project) and the removal of unneeded end user configuration tools. The proxy server is a simple Java application that can be started as follows:

Listing A.7: Proxy start command

```
1        java -jar publish/MVTProxy.jar -p 8080
```

where 8080 has to be be substituted by the desired port.

[1] http://paw-project.sourceforge.net/

The file `conf\reg-filter.xml` has to be adjusted to point to the IP address or hostname of the MVTService (in this example: `heatmaps.no-ip.org`) at the three occurrences in the document. If your servlet container (Tomcat) is running on port other than 80, be sure to add the port to the URL (e.g. `tomcats_base_url:8000`)

Listing A.8: Configuration file `reg_filter.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<reg-filters>
  <reg-filter type="replace" status="active">
    <name>MVTInjection</name>
    <description>Injects the Javascript for Mobile Viewport Tracking</description
        >
    <regexp>
      <match>&lt;\/[ ]*head[ ]*&gt;</match>
      <subst>&lt;script type="text/javascript" src="http://heatmaps.no-ip.org/
          MVTService/viewport.js"&gt;&lt;/script&gt;&lt;script type="text/
          javascript"&gt; MVT.init("heatmaps.no-ip.org"); &lt;/script&gt;&lt;/
          head&gt;</subst>
    </regexp>
  </reg-filter>
  <reg-filter type="replace" status="active">
    <name>MVTUserInjection</name>
    <description>Injects the IFrame for Global Cookies</description>
    <regexp>
      <match>&lt;\/[ ]*body[ ]*&gt;</match>
      <subst>&lt;iframe src="http://heatmaps.no-ip.org/MVTService/mvt_user.html"
          height="1"  width ="1" style="visibility:hidden; top:-10000; left
          :-10000;" id="mvt_iframe"/&gt;&lt;/body&gt;</subst>
    </regexp>
  </reg-filter>
</reg-filters>
```

- *MVTPersistence*: The MVTPersistence project is a decoupled library for persistence tasks using the OR mapper hibernate and business-logic like heatmap drawing and action type determining. Its configuration is based on the `persistence.xml` file for hibernate. The configuration is similar to other hibernate based projects [2] . Main configuration properties are the `hibernate.dialect`, the `hibernate.connection.url`, the `hibernate.connection.username` and the `hibernate.connection.password`.

Listing A.9: Hibernate configuration file `persistence.xml`

```
        <?xml version='1.0' encoding='utf-8'?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://
                 java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
             version="2.0">
   <persistence-unit name="mvtManager" transaction-type="RESOURCE_LOCAL">
      <provider>org.hibernate.ejb.HibernatePersistence</provider>
      <class>at.tuwien.viewport.persistence.entities.ViewportEntry</class>
      <class>at.tuwien.viewport.persistence.entities.Heatmap</class>
```

[2] `http://docs.jboss.org/jbossas/docs/Server_Configuration_Guide/4/html/ch01s02s01.html`

```
10      <class>at.tuwien.viewport.persistence.entities.ImportanceMatrix</class>
11      <class>at.tuwien.viewport.persistence.entities.Image</class>
12      <properties>
13        <property name="hibernate.cache.use_query_cache" value="no"/>
14                <property name="hibernate.cache.use_second_level_cache" value="no
                      "/>
15                <property name="cache.provider_class" value="org.hibernate.cache.
                      NoCacheProvider"/>
16        <property name="hibernate.dialect" value="org.hibernate.dialect.
              MySQLDialect"/>
17        <property name="hibernate.hbm2ddl.auto" value="update"/>
18                <property name="hibernate.connection.driver_class" value="com.
                      mysql.jdbc.Driver"/>
19                <property name="hibernate.connection.url" value="jdbc:mysql://
                      localhost:3306/mvt?autoReconnect=true"/>
20                <property name="hibernate.connection.username" value="mvtuser"/>
21                <property name="hibernate.connection.password" value="
                      some_secret_password"/>
22                <property name="hibernate.connection.pool_size" value="10"/>
23      </properties>
24    </persistence-unit>
25  </persistence>
```

The screenshot component needs some additional configuration when it should be used (e.g. from MVTService) - mainly the paths to the browsers that should be used. The current implementation supports Firefox and Google Chrome. The `browser` property is used to select the browser (*FF* for Mozilla Firefox, *Chrome* for Google Chrome). When using Google Chrome the Selenium Chrome Driver[3] has to be in the same directory as the browsers executable. The `screenfolder` is the path where the screenshots are stored. `screenshotOnVisit` should be set to *true* if a screenshot should be taken for every page visit. If it is set to *false*, screenshots are created only when a new heatmap is created. These screenshots are stored in the database.

Listing A.10: The `config.properties` configuration file

```
1  # Use forward slashes
2  screenfolder = C:/MVT/Images/
3  firefoxExe = C:/MVT/Browser/Firefox15/firefox.exe
4  chromePath = C:/MVT/Browser/chrome/
5  # "FF" or "Chrome"
6  browser = Chrome
7  heatmapFolder = C:/MVT/Heatmaps/
8  heatmapUserSessionFolder = C:/MVT/UserSessions/
9  screenshotOnVisit = true
```

- *MVTService*: The MVTService is based on the MVTPersistence and exposes some Web Service endpoints like the one the injected JavaScript is sending the user actions to. It has to be deployed as a WAR[4]-file on a servlet container like Apache Tomcat. The configuration is done via the contained MVTPersistence.

[3] `https://code.google.com/p/selenium/wiki/ChromeDriver`

[4] Web Module - `http://docs.oracle.com/javaee/5/tutorial/doc/bnadx.html`

- *MVTRegistration*: The MVT Registration page is a decoupled part of the MVTAnlysis application. Its purpose is to allow users the registration of their username and the creation of the global tracking cookie for iOS devices. No configuration is needed as the configuration of the included MVTPersistence is used. This component is not needed in all cases as the same functionality is provided via the MVTAnalysis application.
- *MVTAnalysis*: The MVT Analysis application which was developed as a simple analysis tool for case studies does not need its own configuration as the configuration of the included MVTPersistence is used. It is dependent on the user roles of the servlet container (e.g Apache Tomcat). For normal access, the user should have the *webapp* role, for admin rights, the *webappadmin* role should be given.

Listing A.11: Tomcat configuration file `tomcat_users.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<tomcat-users>
        <role rolename="webapp" />
        <role rolename="webappadmin" />
        <user username="exin" password="some_secret_password" roles="webapp" />
        <user username="admin" password="some_secret_password" roles="webapp,
            webappadmin" />
</tomcat-users>
```

A.2.3 Build and Run Mobile Viewport Tracking

To build and run the project from the source code, a small guide is given:

1. *Check-out from source code management system*: The source code management system used is Mercurial. Pull and update from the repository. Each of the projects uses its own repository. For each project needed, run:

   Listing A.12: Check out using Mercurial Console

   ```
   hg init
   hg pull pathToRepository
   hg update
   ```

   to get a working copy of the project.

2. *Import into Eclipse*: For development, the Eclipse IDE was used. Import the projects into an Eclipse workspace by right clicking the Project Explorer and selecting *Import -> Import -> General -> Existing Projects into Workspace*. Select browse and browse to the the root directory of the projects. Select all projects that should be imported. Optionally, select *copy projects into workspace*, then select *Finish*.

3. *Select the targeted runtimes*: If your target runtime is not called Apache Tomcat v7.0, then you have to set the target runtime to the servlet container you are using by right clicking the project and selecting *Properties -> Targeted Runtimes* and selecting the runtime of your choice. It is recommended to use Apache Tomcat 7.0.

4. *Fix project references*: If the project folders were not renamed after checkout - the dependencies between the projects are set correctly. If project folders were renamed, please be aware that the MVTService and MVTAnalysis projects need the MVTPersistence project on their build path. Set it via right click on the project -> *Properties* -> *Java Build Path* -> *Projects* -> *Add* and selecting MVTPersistence and -> *Properties* -> *Deployment Assembly Add* and again selecting MVTPersistence. Store via *OK*.

5. *Download the Vaadin Plugin*: If not already installed, download the Vaadin Plugin via *Eclipse Marketplace* or *Install New Software* and entering the repository url `http://vaadin.com/eclipse`. This step is optional but the plugin helps updating the Vaadin / GWT version. The current versions of libraries are included in the repository. This only has an impact on Vaadin based projects (MVTAnalysis and MVTRegistration).

6. *Configure Projects*: Follow the instructions in section A.2.2 to configure the applications. Be sure to set the database credentials correctly. The database schema is automatically created if it does not exist.

7. *Build using Eclipse*: All projects can be build via Eclipse. The JAR-file of the MVTProxy can also be build with Apache Ant as a build script is provided. For the Web applications, this is optional as the projects will be built automatically when exporting.

8. *Export*: Export MVTAnalysis, MVTRegistration and MVTService as WAR files via right-click on the project -> *Export* ->*War file*.

9. *Deploy*: Deploy the WAR files via Tomcats Admin Web Page or via Tomcats file system watcher. Deploy the MVTProxy by copying its JAR-file on the server and starting it via a command shell by running

Listing A.13: Proxy start command

```
1 java -jar publish/MVTProxy.jar -p 8080
```

10. *Use*: Configure the smartphones as described in A.2.1. The MVTAnalysis application can be found at `http://tomcats_base_url:tomcat_port/MVTAnalysis`, the registration application can be found at `http://tomcats_base_url:tomcat_port/MVTRegistration`.

The server firewall has to forward communication on the ports of the servlet container - this is typically port 80 - and the port of the proxy server - 8080 in the default configuration. Additionally, outgoing requests have to be permitted for the screenshot component - Mozilla Firefox or Google Chrome.

Bibliography

[1] Massimiliano Albanese, Antonio Picariello, Carlo Sansone, and Lucio Sansone. Web personalization based on static information and dynamic user behavior. In *Proceedings of the 6th annual ACM international workshop on Web information and data management - WIDM '04*, page 80, New York, New York, USA, November 2004. ACM Press.

[2] Tim Berners-Lee. Semantic Web Road map, 1998.

[3] Christian Bizer, T Heath, and T Berners-Lee. Linked data-the story so far. *International Journal on Semantic ...*, 2009.

[4] Marco Brambilla and Piero Fraternali. Large-scale Model-Driven Engineering of web user interaction: The WebML and WebRatio experience. *Science of Computer Programming*, pages 1–35, April 2013.

[5] R. Burget. Layout Based Information Extraction from HTML Documents. In *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007) Vol 2*, volume 2, pages 624–628. IEEE, September 2007.

[6] Georg Buscher, Ralf Biedert, Daniel Heinesch, and Andreas Dengel. Eye tracking analysis of preferred reading regions on the screen. *Proceedings of the 28th of the international conference extended abstracts on Human factors in computing systems - CHI EA '10*, page 3307, 2010.

[7] Deng Cai, Shipeng Yu, JR Wen, and WY Ma. Extracting content structure for web pages based on visual representation. *Web Technologies and Applications*, 2003.

[8] Deng Cai, Shipeng Yu, JR Wen, and WY Ma. VIPS: a visionbased page segmentation algorithm. 2003.

[9] Deepayan Chakrabarti, Ravi Kumar, and Kunal Punera. A graph-theoretic approach to webpage segmentation. *Proceeding of the 17th international conference on World Wide Web - WWW '08*, page 377, 2008.

[10] MC Chen, JR Anderson, and MH Sohn. What can a mouse cursor tell us more?: correlation of eye/mouse movements on web browsing. *CHI'01 extended abstracts on ...*, pages 281–282, 2001.

[11] Krzysztof Czarnecki and Simon Helsen. Classification of model transformation approaches. In *Proceedings of the 2nd OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture*, pages 1–17, 2003.

[12] Alex J. DeWitt. Examining the Order Effect of Website Navigation Menus With Eye Tracking. *Journal of Usability Studies*, 6(1):39–47, November 2010.

[13] Chiara Di Francescomarino, Alessandro Marchetto, and Paolo Tonella. Reverse Engineering of Business Processes exposed as Web Applications. *2009 13th European Conference on Software Maintenance and Reengineering*, pages 139–148, 2009.

[14] Giuseppe Antonio Di Lucca, Anna Rita Fasolino, and Porfirio Tramontana. Reverse engineering Web applications: the WARE approach. *Journal of Software Maintenance and Evolution: Research and Practice*, 16(12):71–101, January 2004.

[15] Cristian Duda, Gianni Frey, Donald Kossmann, Reto Matter, and Chong Zhou. AJAX Crawl: Making AJAX Applications Searchable. *2009 IEEE 25th International Conference on Data Engineering*, pages 78–89, March 2009.

[16] J Fürnkranz. Web Structure Mining Exploiting the Graph Structure of the World-Wide Web. pages 1–10, 2002.

[17] Archana Ganapathi and Steve Zhang. Web analytics and the art of data summarization. In *Managing Large-scale Systems via the Analysis of System Logs and the Application of Machine Learning Techniques on - SLAML '11*, pages 1–9, New York, New York, USA, October 2011. ACM Press.

[18] Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *MIS Quarterly*, 28(1):75–105, March 2004.

[19] Garrett J J. Ajax: A New Approach to Web Applications. http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications, 2005.

[20] S Josephson and ME Holmes. Visual attention to repeated internet images: testing the scanpath theory on the world wide web. *Proceedings of the 2002 symposium on Eye Tracking*, 0535(March), 2002.

[21] M Khoo, Joe Pagano, and AL Washington. Using web metrics to analyze digital libraries. *Proceedings of the 8th ACM/IEEE-CS joint conference on Digital libraries*, pages 375–384, 2008.

[22] Christian Kohlschütter, Peter Fankhauser, and Wolfgang Nejdl. Boilerplate detection using shallow text features. *Proceedings of the third ACM international conference on Web search and data mining - WSDM '10*, page 441, 2010.

[23] Christian Kohlschütter and Wolfgang Nejdl. A Densitometric Approach to Web Page Segmentation Segmentation as a Visual Problem. In *Proceeding CIKM '08 Proceedings of the 17th ACM conference on Information and knowledge management*, pages 1173–1182, 2008.

[24] Bernhard Krüpl-Sypien, Ruslan R. Fayzrakhmanov, Wolfgang Holzinger, Mathias Panzenböck, and Robert Baumgartner. A versatile model for web page representation, information extraction and content re-packaging. In *Proceedings of the 11th ACM symposium on Document engineering - DocEng '11*, pages 129–138. ACM Press, September 2011.

[25] Bernhard Krüpl-Sypien, Ruslan R. Fayzrakhmanov, Wolfgang Holzinger, Mathias Panzenböck, and Robert Baumgartner. A versatile model for web page representation, information extraction and content re-packaging. In *Proceedings of the 11th ACM symposium on Document engineering - DocEng '11*, page 129, New York, New York, USA, September 2011. ACM Press.

[26] Kin Lane. From Web to a Programmable Web to a Programmable World, 2013.

[27] Clicktale Ltd. Homepage. http://www.clicktale.com/, 2013.

[28] a. Marchetto, P. Tonella, and F. Ricca. ReAjax: a reverse engineering tool for Ajax Web applications. *IET Software*, 6(1):33, 2012.

[29] A. Memon, I. Banerjee, and A. Nagarajan. GUI ripping: reverse engineering of graphical user interfaces for testing. *10th Working Conference on Reverse Engineering, 2003. WCRE 2003. Proceedings.*, pages 260–269, 2003.

[30] Ali Mesbah, Engin Bozdag, and Arie Van Deursen. Crawling AJAX by Inferring User Interface State Changes. *2008 Eighth International Conference on Web Engineering*, pages 122–134, July 2008.

[31] JL Meunier. Optimized XY-cut for determining a page reading order. In *Document Analysis and Recognition, 2005. Proceedings. Eighth International Conference on*, pages 347–351. IEEE, 2005.

[32] Network Working Group and Douglas Crockford. JavaScript Object Notation (JSON). http://www.ietf.org/rfc/rfc4627.txt, 2006.

[33] Cristóbal Arellano Oscar Díaz and Gorka Puente. *Wikipedia Customization through Web Augmentation Techniques*. 2012.

[34] Bing Pan, Helene A. Hembrooke, Geri K. Gay, Laura A. Granka, Matthew K. Feusner, and Jill K. Newman. The determinants of web page viewing behavior. In *Proceedings of the Eye tracking research & applications symposium on Eye tracking research & applications - ETRA'2004*, pages 147–154, New York, New York, USA, March 2004. ACM Press.

[35] Beatriz Plaza. Google Analytics for measuring website performance. *Tourism Management*, 32(3):477–481, June 2011.

[36] Gergely Rakoczi. Cast your eyes on Moodle: An eye tracking study investigating learning with Moodle. *... of the 4th International Conference Moodle. si*, (May):203–213, 2010.

[37] Steve Ranger. CIOs predict the death of the PC and the desk phone, 2012.

[38] Ruslan R. Fayzrakhmanov. WPPS: A Novel And Comprehensive Framework For Web Page Understanding And Information Extraction. In *In Proceedings of the IADIS International Conference WWW/Internet 2012 (ICWI 2012)*, pages 19–26. IADIS Press, 2012.

[39] Andrés Sanoja and S Gançarski. Yet Another Hybrid Segmentation Tool. 1(270137):2009–2010, 2012.

[40] K Smith-Gratto and MM Fisher. Gestalt theory: a foundation for instructional screen design. *Journal of Educational Technology Systems*, 27(4):361–371, 1999.

[41] Jaideep Srivastava, Robert Cooley, Mukund Deshpande, and Pang-Ning Tan. Web usage mining. In *ACM SIGKDD Explorations Newsletter*, volume 1, page 12, January 2000.

[42] S. Tilley. Evaluating the reverse engineering capabilities of Web tools for understanding site content and structure: a case study. *Proceedings of the 23rd International Conference on Software Engineering. ICSE 2001*, pages 514–523, 2001.

[43] D Todorovic. Gestalt principles. 3(12):5345, 2008.

[44] W3C. Document Object Model (DOM). http://www.w3.org/DOM/, 2005.

[45] Xin Yang and Yuanchun Shi. Enhanced Gestalt Theory Guided Web Page Segmentation for Mobile Browsing. In *2009 IEEEWICACM International Joint Conference on Web Intelligence and Intelligent Agent Technology*, volume 3, pages 46–49. IEEE, 2009.

[46] Nicholas C. Zakas. The evolution of web development for mobile devices. *Communications of the ACM*, 56(4):42, April 2013.

List of Figures