Tracking User Actions for the Web-Based Front End of ExplorViz

Bachelor's Thesis

Maria Kosche

September 28, 2013

Kiel University
Department of Computer Science
Software Engineering Group

Advised by: Prof. Dr. Wilhelm Hasselbring
M.Sc. Florian Fittkau

Abstract

User tracking is a useful method for locating usability problems. The topic of this thesis is the tracking of user actions in the web-based front end of ExplorViz, a visualisation tool for software landscapes. It is written with the Google Web Toolkit and its architecture is the client-server model. Evaluated tools such as Google Analytics and Piwik require the usage of JavaScript and are therefore not directly compatible with GWT. Evaluated instrumentation frameworks such as Tiny AOP and GIN lack actuality and are not compatible with current versions of GWT.

To develop a tracking method for ExplorViz a simple but working method was used. Tracking functions were integrated directly into the code and the GWT remote procedure call technique was used for passing tracking records from the client to the server. The tracking records are stored in a CSV formatted file for further processing. The evaluation shows that the developed tracking method covers all requirements. Future work includes the auto-generated visualisation of the tracking records and the development of an instrumentation framework for GWT.

Contents

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Goals . . . 1
  - 1.2.1 G1: Identify Existing Tracking Tools and Methods . . . 1
  - 1.2.2 G2: Evaluation of Identified Tracking Tools and Methods . . . 2
  - 1.2.3 G3: Implementation of a New or Existing Tracking Method . . . 2
- 1.3 Document Structure . . . 2

2 Foundations and Technologies **3**
- 2.1 Visualisation . . . 3
  - 2.1.1 Markov Chains . . . 3
  - 2.1.2 Web Graphs . . . 4
- 2.2 Aspect-Oriented Programming . . . 4
- 2.3 WebGL . . . 6
- 2.4 GWT . . . 7
  - 2.4.1 Remote Procedure Call . . . 7
- 2.5 Xtend . . . 8

3 Dynamic Analysis **9**
- 3.1 Tracking . . . 9
- 3.2 Monitoring . . . 9
  - 3.2.1 Kieker . . . 10
- 3.3 ExplorViz . . . 10
- 3.4 Experiments . . . 12
  - 3.4.1 Threats to Validity . . . 12

4 Evaluation of Existing Tracking Tools and Methods **15**
- 4.1 Assessment Criteria . . . 15
- 4.2 Google Analytics . . . 16
- 4.3 Piwik . . . 17
- 4.4 Instrumentation Frameworks . . . 18
  - 4.4.1 GWT ENT and Tiny AOP . . . 18
  - 4.4.2 GIN . . . 18
  - 4.4.3 Spring . . . 19
- 4.5 Direct . . . 19
- 4.6 Overall Results . . . 20

5 Development of a Tracking Method in ExplorViz **21**
5.1 Goals of Tracking 21
5.2 Approach 21
5.3 Activities 25

6 Evaluation of the Developed Tracking Tool **27**
6.1 Completeness Evaluation 27
6.1.1 Quality Criteria 27
6.1.2 Scenarios 28
6.1.3 Results 29
6.1.4 Discussion 32
6.1.5 Threats to Validity 32

7 Conclusions and Future Work **35**
7.1 Related Work 35
7.2 Conclusions 35
7.3 Future Work 36

Bibliography **37**

A ExplorViz DVD **41**

List of Figures

2.1 Simple Markov chain example 4
2.2 Web graph layers (taken from Chen et al. 2004) 5
a Vertex size is proportional to number of visitors 5
b Edge thickness gives information about amount of link usage 5
c Vertex colour stands for access time per pages 5
d Edge colour represents access probability of links 5
2.3 RPC architecture (taken from GWTProject 2013) 8

3.1 Overview of Kieker (taken from Kieker 2013) 10
3.2 Landscape viewer in ExplorViz 11
3.3 System viewer in ExplorViz (taken from Fittkau et al. 2013) 11
a System viewer with closed `service` package 11
b System viewer with opened `service` package 11
3.4 Code viewer in ExplorViz 12

4.1 Google Analytics dashboard (taken from Google 2013) 16

5.1 Overview of the important packages 22
5.2 Class diagram of user tracking records 23
5.3 RPC class structure for the user tracking 24
5.4 Sequence for double clicking a node group 26

6.1 Activity diagram for scenario S1 in the landscape level view 28
6.2 Activity diagram for scenario S2 in the system level view 29
6.3 Activity diagram for scenario S3 in the code viewer 29
6.4 Possible Markov chain for the landscape viewer 30

Chapter 1

Introduction

1.1 Motivation

Usability tests are a necessity in the development of applications. They help to increase the competitive capability. Developers often have difficulties thinking like the users of the software in development. Since they are intensively employed with their software, it is hard for them to imagine what problems inexperienced users can have. Therefor, it is very important to understand the user's position. Tracking technologies are a useful method for this purpose. They investigate the user's behaviour by recording user actions, especially the user's mouse and keyboard interaction on desktop systems. They can also be used for logging the time it takes to perform certain tasks. The information can be used for locating usability problems. Subsequently, it is possible to optimise the structure and functionality of an application based on tracking data.

This raises the question what tracking methods and tools already exist, how they work, to what degree they are compatible with existing software, and how to integrate a new or existing tracking method or tool into a given application.

The concern of this thesis will be answering these questions using the example of tracking user actions in the web-based front end of ExplorViz.

1.2 Goals

The main objective is an extension for the web application ExplorViz. User actions shall be tracked and therefor, we define the following goals:

1.2.1 G1: Identify Existing Tracking Tools and Methods

First of all a survey of tracking tools and methods is required. This is useful for an orientation and it is of course not recommended to implement a completely new tracking method if there already exist some valuable solutions.

1.2.2 G2: Evaluation of Identified Tracking Tools and Methods

It is not sufficient to only identify existing tracking tools and methods. In a next step they are evaluated in terms of a possible adaption for ExplorViz. Even if there is no appropriate tracking tool or method, the evaluation is a facility for imagining possible features and methods.

1.2.3 G3: Implementation of a New or Existing Tracking Method

The final step is the implementation of a tracking method in ExplorViz. Obviously, the development depends on the results of the preceding evaluation. The goal is a functioning implementation for tracking the user mouse actions in the landscape, system, and code viewers of ExplorViz, which are described in Section 3.3.

1.3 Document Structure

Chapter 2 describes the technologies which are used and investigated for the thesis. Thereafter, Chapter 3 attends to the main topics like tracking, monitoring, and the experimental aspect of tracking. Afterwards, Chapter 4 provides an overview of the existing technologies and contains the evaluation of the investigated tools and methods. Chapter 5 shows how the development proceeded and which steps were made. In Chapter 6 the evaluation of the implemented tracking method is conducted and finally, future work is described and conclusions are drawn in Chapter 7.

Chapter 2

Foundations and Technologies

This chapter gives an overview of foundations and technologies which are used in the thesis.

2.1 Visualisation

For analysing tracking and monitoring data, it is recommended to use an appropriate visualisation. Therefor, two possibilities will be introduced. First, Markov chains are useful for visualising states. They concentrate only on the probabilities of moving from one state to another, but have many use cases, because defining states is flexible and can adapt to the purpose. Furthermore, they are defined mathematically and can be easily examined, analysed, and processed. Second, web graphs are created directly for web applications and can present a bigger amount of information. But they are also more abstract and therefore, more complex.

2.1.1 Markov Chains

Markov chains are stochastic processes often visualised by diagrams. There are two types of Markov chains: discrete-time and continuous-time ones. Since we only deal with web applications with a finite number of pages respectively states, we will only investigate discrete-time Markov chains. The purpose of Markov chains is to give probabilities of events occurring. The particularity is that they retain no memory, meaning that prognoses made by Markov chains only depend on the current state and not on the past. This way there is no need for recording the history for calculating the probability of a next state.

Figure 2.1 shows a simple Markov chain diagram. The edge labels describe the probability of the next state. Starting from state s_0 there is one path to s_2 with probability 1. State s_1 leads either to s_0 or to s_2 with equal probability $1/2$. Finally, the last path moves from state s_2 to s_1 with probability $1/3$. Since the total probability of all transitions leaving a given state must equal 1, we can infer that we stay in s_2 with probability $2/3$, even though this transition is not explicitly drawn in the diagram. Imagining each state represents a view, a website, or the state of several objects, tracking data can be visualised using Markov chains to illustrate the behaviour of users using a web application. [Norris 1998]

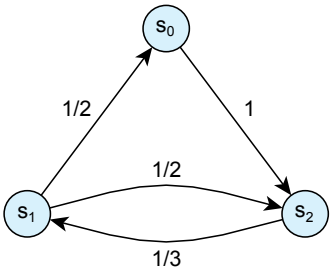


Figure 2.1. Simple Markov chain example

2.1.2 Web Graphs

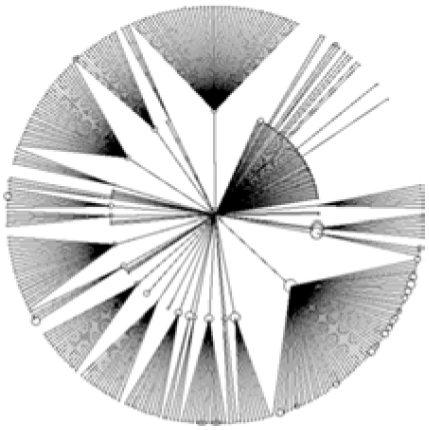
Another possibility for visualising the behaviour of users on web applications are web graphs. A tree structure is used to represent a whole website. Figure 2.2 shows an example of the different layers used in a web graph. These layers are also called web images. Nodes represent pages and edges symbolise hyperlinks between them. Figure 2.2a emphasises the number of visitors per page by varying the vertex size accordingly. Figure 2.2b stresses the edge thickness which is interpreted as the link usage. Figure 2.2c displays the layer of vertex colours which correspond to the access time per page. Figure 2.2d finally completes the layers with edge colours indicating the access probability of links. [Chen et al. 2004]

2.2 Aspect-Oriented Programming

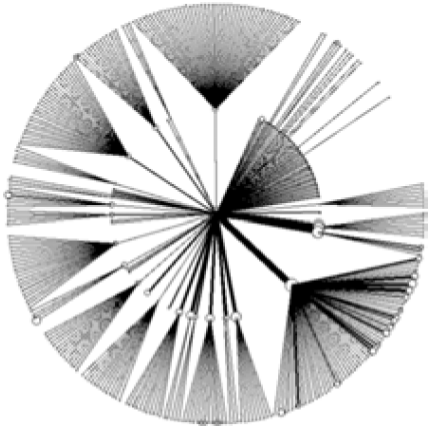
Aspect-oriented programming (AOP) is used for separating certain functionality from the business logic code. Code can be of core level concern. This is functionality of code which can be easily modularised in several classes and functions. System level concern or crosscutting concern, however, concerns the whole system or must be implemented in functions which actually do something else. Logging is an example of a system level concern. It would be implemented in several functions, although it has nothing directly to do with the functions. In this case aspects are appropriate.

For marking points in the code where some functionality like logging should be added, so-called join points are defined. These are often just annotations which are added to functions. Pointcuts are now implemented for matching the relevant join point to the right aspect. The aspect contains advices describing what to do when a pointcut has matched a join point. For example, it is possible to execute some logging code before continuing with the execution of the program. Very popular for aspect-oriented programming in Java is AspectJ.

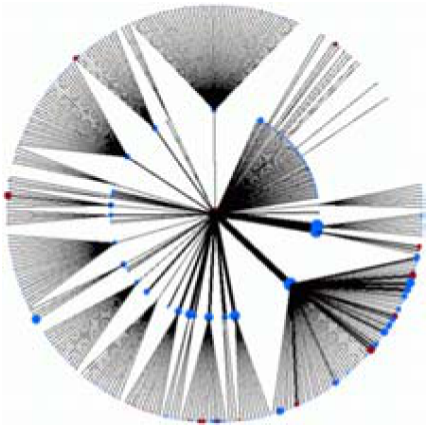
Listing 2.1 shows a small example of a method which shall be logged. The logging code is directly implemented in `testMethod`. Now aspect-oriented programming will



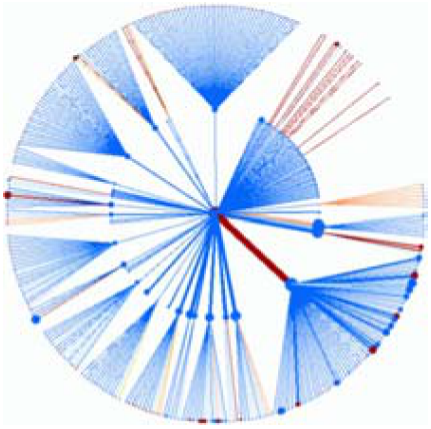
(a) Vertex size is proportional to number of visitors



(b) Edge thickness gives information about amount of link usage



(c) Vertex colour stands for access time per pages



(d) Edge colour represents access probability of links

Figure 2.2. Web graph layers (taken from Chen et al. 2004)

```
1  public void testMethod() {
2          logger.trace("Entering testMethod");
3
4          // executing method
5          System.out.println("Hello World!");
6  }
```

Listing 2.1. Logging a method without aspect-oriented programming

```
1  @LogMethod
2  public void testMethod() {
3          // executing method
4          System.out.println("Hello World!");
5  }
6
7  @Aspect
8  public class LoggingAspect {
9
10         @Before("annotation(test.package.LogMethod)")
11         public void logBefore(JoinPoint joinPoint) {
12
13                 logger.trace("Entering " + joinPoint.getSignature().getName());
14         }
15
16 }
```

Listing 2.2. Logging a method with aspect-oriented programming

be used for separating the functionality. A custom annotation `LogMethod` is declared which marks the join point as shown in Listing 2.2. The class `LoggingAspect` is the aspect containing the pointcuts and the advices. In this case the advice `@Before` is used to execute the logging code before the execution of `testMethod`. The pointcut is defined as `annotation(test.package.LogMethod)` meaning that every method with our `LogMethod` annotation will be matched. For example, valid pointcuts could also match every method in a specific class.

2.3 WebGL

WebGL stands for Web Graphics Library and is a JavaScript API used to display 3D graphics in a web browser. It uses the OpenGL shading language GLSL and is based on OpenGL ES 2.0 (ES stands for Embedded Systems) which is the OpenGL specification

version targeted at handheld and embedded devices such as cell phones, PDAs, consoles, and vehicles. In general, OpenGL is a cross-platform standard 3D API for advanced 3D graphics. It is widely accepted and is used by games such as Half-Life, GLTron, Portal, Minecraft, StarCraft II, and World of Goo. Furthermore, applications such as Blender or Google Earth include an OpenGL renderer. [Munshi et al. 2008; Shreiner et al. 2009]

As it became important to provide web-based and real-time rendering, WebGL was specified and it opened up possibilities for web-based 3D environments in web browsers without any plug-in components. According to the distribution of browsers, web-based 3D applications are also available for smart phones, tablets etc. WebGL also offers an integration with HTML content, so that, for instance, the interaction with other HTML elements and the use of standard HTML event handling mechanisms are available. Furthermore, it is possible to use WebGL for 2D graphics as well. [WebGL 2013; Cantor 2012; Anttonen and Salminen 2011; Seidelin 2011]

2.4 GWT

GWT stands for Google Web Toolkit and is a toolkit for developing web applications. It enables the development and implementation of AJAX applications. Its characteristic is a Java to JavaScript compiler which allows developers to create complex JavaScript front end applications in Java.

AJAX is an acronym for "Asynchronous JavaScript and XML" and describes a concept for transmitting data between client and server asynchronously. In this way it is possible to submit HTTP requests and change an HTML site without reloading it. AJAX is platform independent and can be used by every browser which supports JavaScript. Besides, there is no need for installing a plug-in for using AJAX applications.

GWT allows the developer to leverage his knowledge of Java and to use approved development environments such as Eclipse. Furthermore, GWT comes with different panel layouts and many widgets which can be combined into a graphical user interface. Creating an user interface is thus similar to creating it with Swing or SWT, and the application becomes more desktop-like. [Chaganti 2007; Wargolet 2011]

2.4.1 Remote Procedure Call

The GWT remote procedure call (RPC) is a technique for simplifying the exchange of Java objects over HTTP between client and server components. A service on the server-side is invoked from the client, an auto-generated proxy class is used for the communication between server and client, and the serialisation of the Java objects is handled by GWT. Therefor, the interface `Serializable` (Java) respectively `IsSerializable` (GWT) must be implemented by the object classes. Figure 2.3 shows an overview of the RPC architecture. The service proxy class (shown in green) is the core of the communication between client and server. Calls from the client are passed to the correct class on the server-side by the proxy

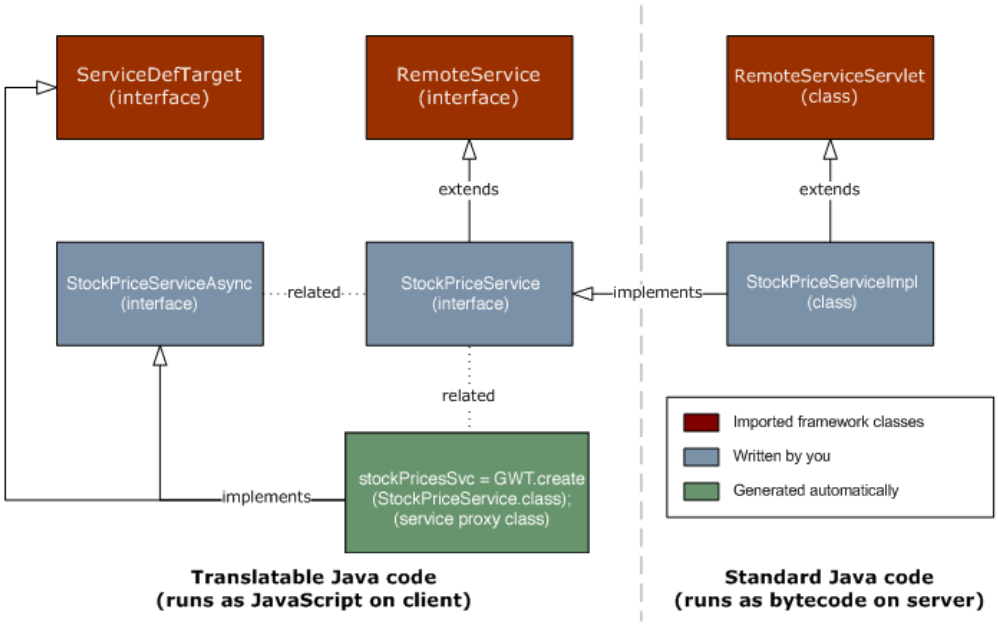


Figure 2.3. RPC architecture (taken from GWTProject 2013)

class. It is automatically generated with the creation of the `StockPriceService`. This is an interface realised by the Service on the server-side. The related `StockPriceServiceAsync` enables catching the return value of the server-side service. [tutorialspoint 2013; GWTProject 2013]

2.5 Xtend

Xtend is a programming language whose syntax is similar to Java but provides more advanced language elements such as closures. It compiles into Java source code and is still object-oriented and imperative, but also integrates some features known from functional programming languages. For example, Xtend supports lambda expressions which can be used for implementing anonymous classes. Programmers who are already familiar with Java are able to leverage their knowledge and are able to quickly learn Xtend. Some syntactical simplifications are made such as unmatched type inference and leaving out semicolons and empty parentheses. Finally, Xtend is completely compatible to Java while being more readable and expressive. [Eclipse 2013; Lübbe 2012]

Chapter 3

Dynamic Analysis

In the following, we describe the foundations of tracking and monitoring. ExplorViz will be introduced in Section 3.3 while Section 3.4 describes the threats to validity concerning web tracking.

3.1 Tracking

Tracking and especially web tracking is used to collect, store, and connect user behaviour records. Tracking is often associated with advertisement companies, which actively collect information such as age, sex, and place of residence about users and accumulate it in user profiles. These profiles are used to show users individualised advertisements instead of random ones. Apart from this, tracking is also used for law enforcement. Tracking technologies enable spying on individuals and solving crimes such as identity theft and credit card fraud. Moreover, a further major motivation for tracking are usability tests of applications. "By observing the steps an individual performs while trying to solve a certain task [...], usability problems can be discovered and fixed." [Schmücker 2011, p.1] It is possible to capture detailed records of user mouse and keyboard input. Recording cursor movement paths can be helpful for identifying problems when locating or using certain functionality. Furthermore, this data can be extended with a time component so that tracking can be used to analyse how long certain tasks take, which tasks cause the main problems, and in which order the user proceeds. In addition "web analytics, a related field, focusses less on the individual user, but more on the performance of a website as a whole" [Schmücker 2011, p.1]. Tracking can help improve the structure of a website by listing, for example, the number of visitors over time, the time visitors spend on one site, and which pages they look at. [Schmücker 2011, section I. - III.] The most popular tools used for web tracking are Google Analytics and Piwik (Open Source). They are investigated in more detail and evaluated in Chapter 4. [Schmücker 2011]

3.2 Monitoring

Tracking is usually used for observing and recording user actions and user behaviour. Monitoring, on the other hand, is used for measuring the performance of an application.

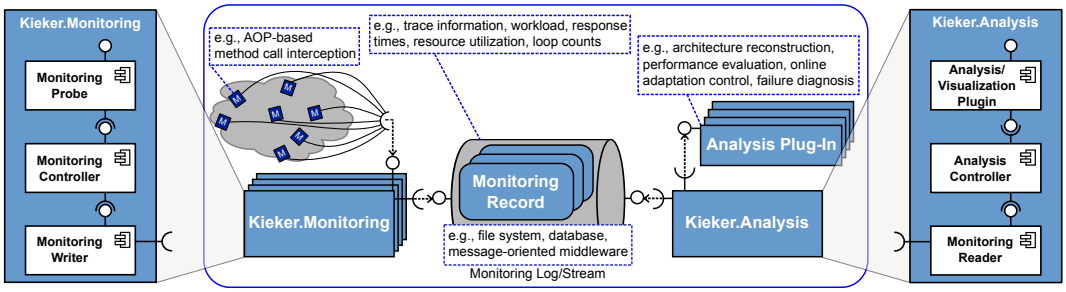


Figure 3.1. Overview of Kieker (taken from Kieker 2013)

3.2.1 Kieker

Kieker is a Java-based tool which provides Application Performance Monitoring (APM). It therefor collects and analyses monitoring data. It focuses on the application's run-time behaviour. On the application and service level the operation response times, the user sessions, and traces are investigated. On the system level the collected data can include the CPU utilisation, memory usage etc. [van Hoorn et al. 2012; 2009; Rohr et al. 2008] Figure 3.1 gives an overview of the features of Kieker. It shows the Kieker monitoring component which uses, for example, AOP-based method call interception. Monitoring records are created and can be analysed by the Kieker analysis component.

3.3 ExplorViz

ExplorViz is an online trace visualisation for large software landscapes which can arise from the high number of software systems a company uses. It is intended as a support for software engineers trying to comprehend software systems, for example, when creating new features. Therefor, it provides the functionality to disclose details such as the communication between programs and the control flow in an application. Using an adequate and comprehensible abstraction for its visualisations, ExplorViz allows to structure the relevant information. For this purpose there are different views to examine the software structure with.

The landscape level view, which is the main view, contains a graph with nodes, node groups, and edges representing the structure of the software system. Figure 3.2 shows an exemplary landscape level view. Nodes contain several applications. These can be investigated in more detail by double clicking. This opens a new three-dimensional view, the system level view, which is related to the model of code cities and is shown in Figure 3.3. Figure 3.3a shows the initial view while Figure 3.3b shows the same view after opening the `service` package via double click. The system level view uses WebGL for the visualisation and a navigation with moving and zooming is possible. By right clicking on components both in the landscape and the system level view, a popup menu is opened which provides

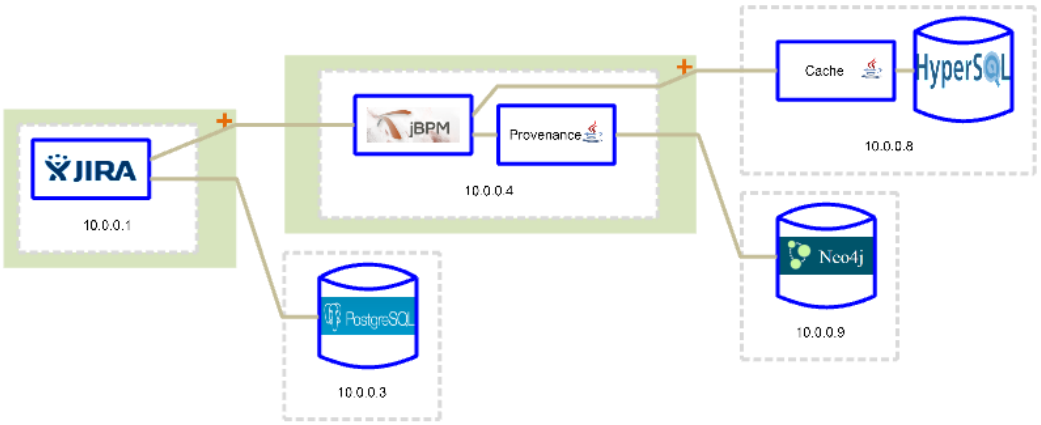
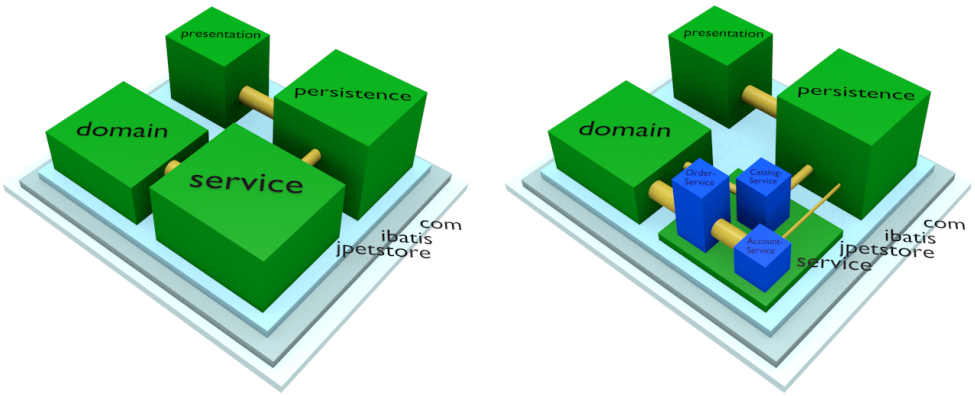


Figure 3.2. Landscape viewer in ExplorViz



(a) System viewer with closed `service` package

(b) System viewer with opened `service` package

Figure 3.3. System viewer in ExplorViz (taken from Fittkau et al. 2013)

access to the code viewer. The code viewer, shown in Figure 3.4, provides a tree diagram which corresponds to the package and class structure of a project. Files of the project can be opened directly.

Besides, performance is an important aspect, so the user experience is not degraded while processing a huge amount of traces.

The technologies described in Section 2.3 through Section 2.5, namely, WebGL, GWT, and Xtend, are used for the development of the ExplorViz web application. [Fittkau et al. 2013]

Figure 3.4. Code viewer in ExplorViz

3.4 Experiments

Tracking user actions can be seen as a measurement method for experiments. There are different modalities like explanation of functionality or even the mood of the user. One has to deal with experiments for analysing the results of tracking.

An experiment is "a test under controlled conditions made to either demonstrate a known truth, examine the validity of a hypothesis, or determine the efficacy of something previously untried." [Shadish et al. 2002, p.1] Crucial here is the relationship between cause and effect, called causal relationship. The cause is "the producer of an effect, result, or consequence." [Shadish et al. 2002, p.1] The effect is the difference between what happened under the experimental condition as opposed to what would have happened without the experimental condition (called counterfactual). However, the counterfactual cannot be observed, because it is not possible for a given subject to both receive and not receive a treatment. [Shadish et al. 2002]

3.4.1 Threats to Validity

Researchers aspire to attain generalised causal knowledge, but causal generalisation conflicts with the fact that experiments have a very specific context. Each experiment consists of subjects, of the treatments themselves, of observations made on the subjects, and of the settings in which the experiment is conducted. Shadish defines 4 validity types, namely, internal validity, statistical conclusion validity, external validity, and construct validity.

Internal validity is closely related to statistical conclusion validity. Both focus on the relationship between treatment and conclusion. But since we are more interested in investigating causal-reasoning errors than in statistical inferences, statistical conclusion validity is not described further. External validity and construct validity are related as well, since they are both generalisations. Construct validity, however, focuses on questions about the definition of constructs to be measured, or about the units of measurement. These will already be answered by the identification and evaluation of tracking methods and tools. Therefor, only external validity will be described in more detail. [Shadish et al. 2002]

Internal Validity

Internal validity refers to the degree to which a causal conclusion is warranted [Johnson 1997]. Below some reasons are given for questioning the internal validity that are connected to web tracking.

Ambiguous temporal precedence There is uncertainty which variable is the cause and which is the effect.

Selection Differences between subject groups which may interact with the observed effect.

History A third variable may have influence on the observed effect.

Testing Repeatedly testing may influence the results, because practise and familiarity are relevant mechanisms.

External Validity

External validity refers to the degree to which experiments respectively their results are generalisable.

Interaction of the Causal Relationship with Subjects Subjects may have properties that interact with variables to analyse.

Interaction of the Causal Relationship Over Treatment Variations Different treatments may cause different effects.

Interaction of the Causal Relationship with Settings Environmental factors such as time, location etc. may affect the results.

Context-Dependent Mediation If the explanation differs from time to time, the results may also differ.

Chapter 4

Evaluation of Existing Tracking Tools and Methods

	Google Analytics	Piwik	Tiny AOP	GWT ENT	Guice/ GIN	Spring	Direct
Completeness	✓	–	✓	✓	✓	✓	✓
Adaptability	–	–	✓	✓	✓	✓	✓
Usability	✓	✓	✗	✗	–	–	✓
Actuality	✓	✓	✗	✗	✗	✗	✓
Development Activity	✓	✓	✗	✗	✓	–	✓

Legend: ✓ given – partially given ✗ not given

Table 4.1. Overview of evaluation results

Table 4.1 summarises the evaluation of the different tracking tools and methods. The criteria are described and explained in Section 4.1. Afterwards, the different tools and methods are introduced and analysed in terms of these criteria in Section 4.2 to 4.5. Finally, the results of the evaluation are summarised in Section 4.6.

4.1 Assessment Criteria

The criteria for the evaluation of existing tracking tools and methods are inspired by the Consortium for IT Software Quality. In a regular time interval, specifications for software quality measures are published. [Consortium for IT Software Quality 2012]

The criteria “Adaptability”, “Usability”, and “Actuality” are borrowed from these specifications. Here, they are specified and described for the exact purpose of using them for ExplorViz. Also, we add the criteria “Completeness” and “Development Activity”.

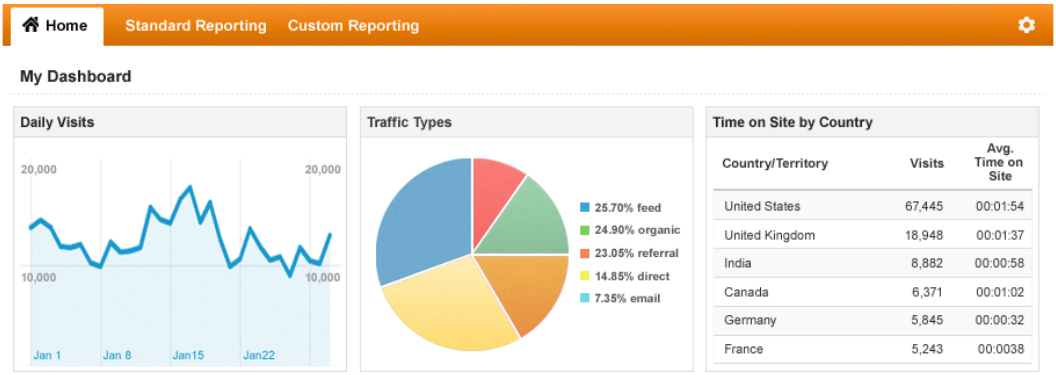


Figure 4.1. Google Analytics dashboard (taken from Google 2013)

Completeness The requirements of tracking user mouse actions in landscape viewer, system viewer, and code viewer should be covered. This includes the tracking of WebGL components.

Adaptability The way of integrating the functionality should be compatible to the existing software structure and configuration.

Usability The documentation is part of this criterion. The tracking method should be documented sufficiently well for understanding and implementing it.

Actuality This criterion describes the actuality of the tools and illustrates whether current or only old versions are supported. It also describes the functioning and stability of the tools and methods.

Development Activity The development activity allows a forecast for the future support and the improvement of the tool and therefore it is important for the sustainability of the criteria above.

4.2 Google Analytics

Google Analytics is a tool for analysing websites. It comes with features like real-time reporting, in-page analysis, visualisation of user paths, and event tracking. A dashboard of Google Analytics could look like the one displayed in Figure 4.1.

The analysis tool measures the number of users visiting a site, offers the opportunity to visualise the paths users take through the application, and provides in-page analysis, that is information about the probability of each link on a page to be clicked. The visualisations generated by Google Analytics are easy to understand and several tools are specialised on finding very particular data among all the recorded data. [Google 2013]

Completeness The event tracking feature provides support for tracking any Flash-driven element such as a flash website or a flash movie player, and embedded AJAX page elements. Since GWT is used for developing AJAX applications as mentioned in Section 2.4, Google Analytics could be used for tracking these elements. Thus, completeness is given.

Adaptability In order to use the functionality of Google Analytics, a registration is needed. Then, code snippets together with an account ID are inserted in the existing code to connect and send usage data to the personal account. The setup and configuration is performed through the account website.

Due to that, Google Analytics is not an appropriate solution for tracking user actions in ExplorViz. Since ExplorViz is developed with GWT and thus in Java, it is cumbersome to place JavaScript snippets in the existing implementation. Furthermore, it is undesirable to store the tracking data at an external service due to data protection concerns. This makes it neither impossible nor recommendable to adapt Google Analytics for ExplorViz.

Usability, Actuality, Development Activity Nevertheless, Google Analytics has a big community and offers considerable support. The tool is up-to-date and actively developed for improving features, compatibility, and usability.

4.3 Piwik

Like Google Analytics, Piwik is a multifunctional tool for tracking and analysing websites. It concentrates on information about the visitors such as how long do they visit the website, from which country they come, and which browser they use. Besides, Piwik offers the possibility to integrate code snippets and event listeners in existing JavaScript code to track several functions.

Completeness Although Piwik provides similar features, its support for asynchronous tracking is not as technically mature as the support provided by Google Analytics. Even so, it covers the completeness requirements, if barely.

Adaptability Piwik provides a GWT wrapper, but for the moment it only supports a subset of the Piwik functionality. Unfortunately, these functions are insufficient and other alternatives involve the integration of JavaScript code. Therefore, the adaptability is rated on par with Google Analytics.

Usability and Actuality Piwik is open-source and lively exchange exists between community and developers. This enables flexibility, quick support, and updates. So the usability and actuality can be rated as given.

Development Activity Further development is planned for the mentioned GWT wrapper. If it gains more features in the future, Piwik will become a good solution for tracking user actions in GWT.

4.4 Instrumentation Frameworks

For the evaluation we investigate instrumentation frameworks for GWT. Since logging is of crosscutting concern, it is recommended to separate it from the core level concern.

4.4.1 GWT ENT and Tiny AOP

First of all we investigate two frameworks for aspect-oriented programming. Since they are very similar, GWT ENT and Tiny AOP are combined in this subsection.

Completeness Both frameworks are used directly in the code and therefore, the logging functionality can be placed wherever necessary. This means all requirements for completeness are covered. GWT ENT provides even more functionality, for example, reflection and data binding, but these features are irrelevant to the tracking of user actions in ExplorViz.

Adaptability Since the frameworks are designed for GWT it is not time-consuming to integrate GWT ENT and Tiny AOP in GWT.

Usability A disadvantage is the usability of the frameworks. They are hardly documented and the documentation mainly consists of example applications which lack further explanations and more specific use cases.

Actuality Unfortunately, the frameworks are outdated and not compatible with the latest GWT version, which is also used in ExplorViz.

Development Activity The latest updates go back to the year 2010 for Tiny AOP and 2011 for GWT ENT. This means that it is unrealistic that these frameworks will be updated in the near future. In summary, it seems as if there is no current development on aspect-oriented programming for GWT at all.

4.4.2 GIN

GIN stands for GWT Injection and is a framework based on Guice, a Google framework for injection. It provides method injections and is therefore similar to aspect-oriented programming.

Completeness Like GWT ENT and Tiny AOP, GIN is used directly in the code. It can thus be flexibly used and hence covers all requirements for this criterion.

Adaptability GIN is an extended version of Guice for GWT. Therefor, the integration in ExplorViz is possible by implication.

Usability Documentation for GIN is not very extensive, but there is an active platform for the exchange between users and developers. Anyhow, the documentation for Guice is more informative and overall the documentation for the GWT adaption is hardly worth mentioning.

Actuality There still exist some issues with using method injection for aspect-oriented programming. There is no release that works for this use case.

Development Activity Although there is no current stable version for the latest GWT versions, the development is still in progress and it is possible that a version supporting aspect-oriented programming in GWT will soon be released.

4.4.3 Spring

Along with the frameworks for GWT, there exists Spring, a well known framework for Java. It provides, besides many other features, aspect-oriented programming.

Completeness Like with the GWT AOP frameworks, the functionality is implemented directly in the code and therefore covers all requirements.

Adaptability ExplorViz is written in Java and for this reason it is possible to integrate Spring with it after adding the dependencies.

Usability Although Spring has a big community, it is rarely used in combination with GWT. The missing documentation and missing support for this purpose makes an implementation impractical.

Actuality Due to the lack of documentation, it seems impossible to combine Spring with ExplorViz and its requirements on user tracking.

Development Activity Spring shows much development activity, but the integration of aspect-oriented programming in GWT received no priority so far.

4.5 Direct

The “direct” method implements the tracking methods directly in the code. It is not the most elegant method, because the logging functionality is not separated from the business logic code, but it is simple and functional.

Completeness Since the mechanism is very similar, there is no big difference to the instrumentation frameworks when considering the completeness.

Adaptability No additional dependencies are needed and the tracking functions can be directly implemented in Java for integrating them in ExplorViz.

Usability Using basic GWT techniques, there is no need of further documentation. The existing documentation of the GWT project is sufficient for implementing the required features.

Actuality and Development Activity The actuality and development activity refers to the development of GWT itself. Therefore, the technique of direct tracking cannot become outdated and is always on the same level as GWT.

4.6 Overall Results

The investigated existing tools and methods were not appropriate for the purpose of logging user actions in ExplorViz. Therefore, the decision was made to simply track the user actions with the "direct" method. With the information given at the specific code points, records will be created which shall contain the relevant tracking information. Using these records the implementation can be easily exchanged in case an instrumentation framework meets the requirements in the future. Finally, the records shall be passed to the server, which will be done with the GWT RPC technique. The explicit implementation will be described in more detail in the next section.

Chapter 5

Development of a Tracking Method in ExplorViz

This chapter describes the goals of and the approach for the development. Furthermore, the developed tracking method is introduced in detail. All of the classes and packages mentioned in this chapter are included on the DVD in Appendix A for easy reference.

5.1 Goals of Tracking

The tracking of user actions in ExplorViz includes the collection of interaction information with the landscape viewer, system viewer, and code viewer. The interaction in the landscape level view will be tracked for each element, including right click and double click actions. The same holds for the system level view. For the code viewer information about opened files will be tracked.

5.2 Approach

A package overview of the implementation is provided in Figure 5.1. On the client-side is the `explorviz.visualization` package. The `explorviz.visualization.engine.usertracking` package contains the functionality of tracking several methods for the two other packages, `explorviz.visualization.codeviewer` and `explorviz.visualization.engine.interaction`. It generates a record for each event. The different record types are defined in the package `explorviz.shared.records` which is shared with the server-side. The client invokes a service on the server-side, which is shown by the connection to the `explorviz.server.usertracking` package.

The service invocation is conducted with the GWT remote procedure call technique as introduced in Section 2.4.1. It is used for transferring the records from client to server. Afterwards, the records can be written to a CSV file which will be the log file for the tracking data. Therefor, each record implements a method `csvSerialize` which formats the data for writing it to the log file. Each line in the file contains a time stamp in milliseconds at the first position and the simple class name of the created record at the second position. The name of the record, such as “NodeGroupOpenRecord” or “ApplicationOpenPopupMenuRecord”,

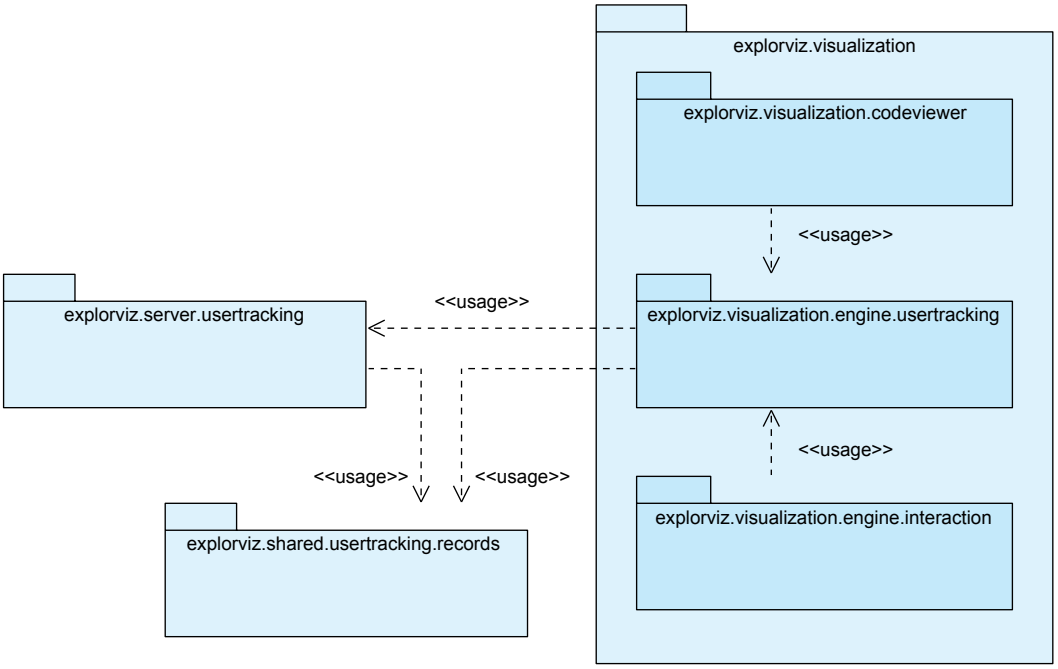


Figure 5.1. Overview of the important packages

provides information about the performed action. Afterwards, the properties of the object which was interacted with are logged.

The records used for the tracking are listed in Figure 5.2. Records that describe the general objects, such as `NodeGroupRecord` or `ApplicationRecord`, extend the abstract class `UsertrackingRecord`. They contain general properties for their particular type of record. The subclasses of these records add information about the type of interaction, for example `NodeGroupOpenRecord` or `NodeGroupCloseRecord`. The `csvSerialize` method of the super class returns a string with the general properties of the record in CSV format using ";" as delimiter. The subclasses add the specific description to the CSV string. The result is then used for the log file.

Due to the remote procedure call technique of GWT, the records are serialised by GWT. Therefor, the abstract class `UsertrackingRecord` implements the GWT interface `IsSerializable`. Other requirements for user defined classes to be serialisable include a default constructor and the fact that every non-final field is itself serialisable. By default all the primitive types, their wrapper objects, and arrays of serialisable types are serialisable.

To describe the RPC methodology further, Figure 5.3 shows the implemented or modified classes in addition to the RPC overview in Section 2.4.1. The `Usertracking` class triggers the tracking process by creating the service proxy class and executing the

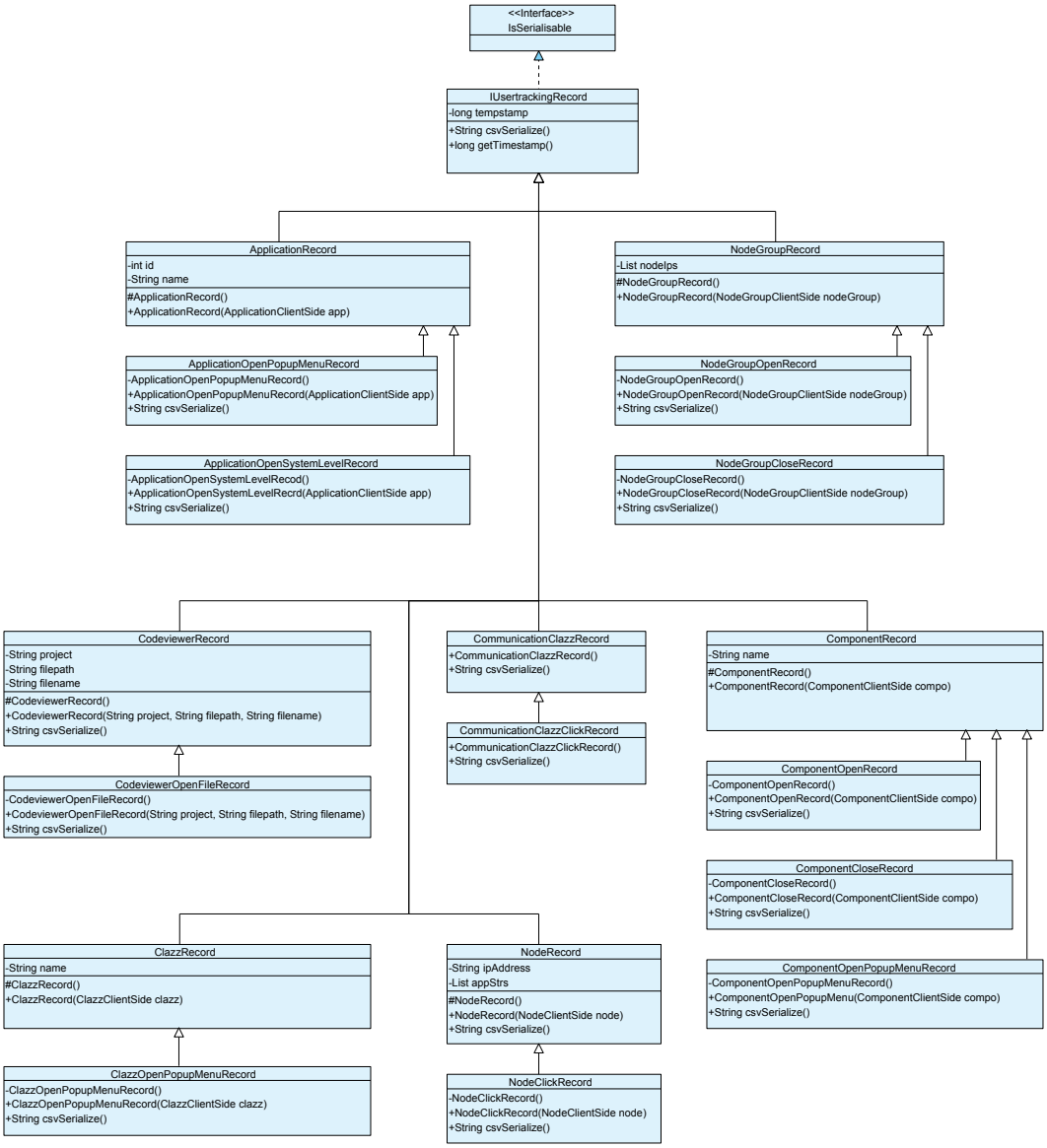


Figure 5.2. Class diagram of user tracking records

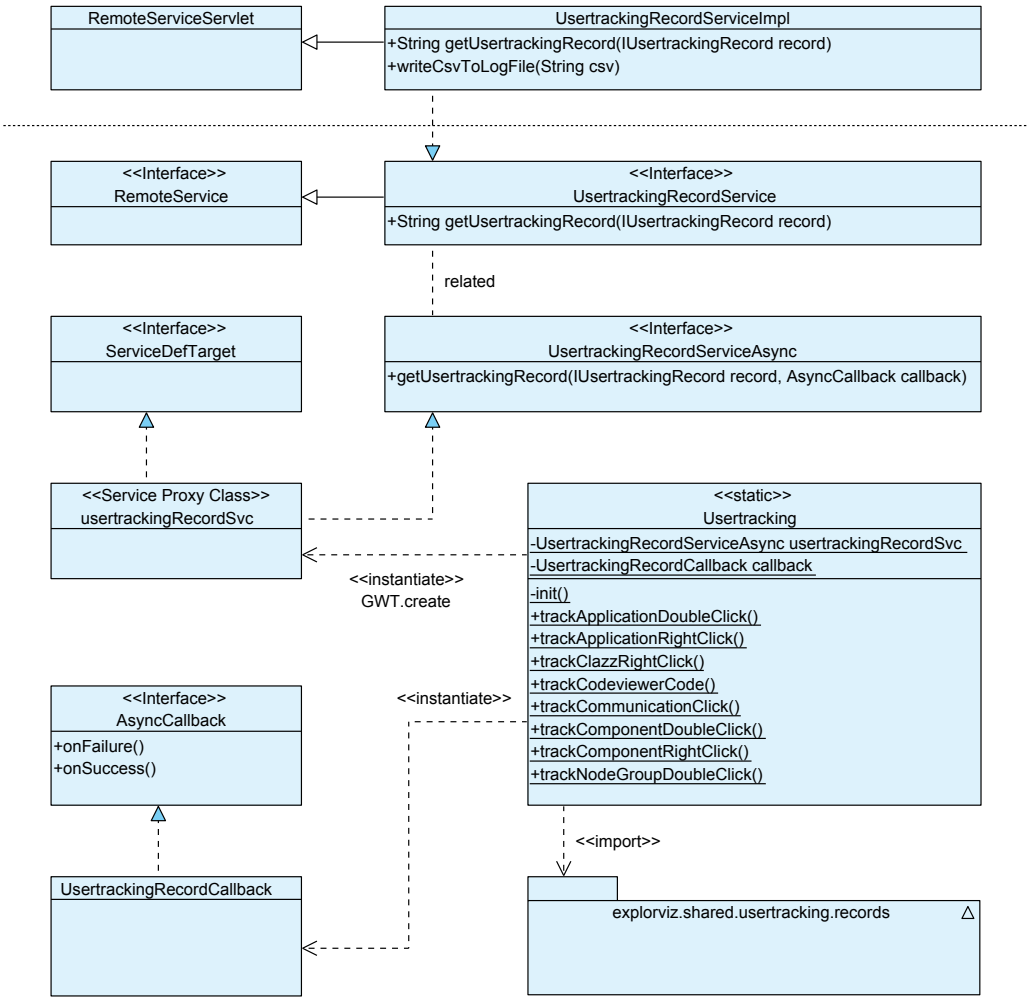


Figure 5.3. RPC class structure for the user tracking

getUsertrackingRecord method. To that end, an AsyncCallback field is instantiated. The UsertrackingRecordCallback<String> class realises the AsyncCallback<T> interface and defines the behaviour depending on the return value of the remote procedure call. Since the return value is not important to us, it is only used for exception handling. Furthermore, the records are created by the Usertracking class and then serialised by GWT for passing them to the server. In the UsertrackingRecordServiceImpl class implemented on the server-side, the deserialised records are collected and serialised for the log file using the csvSerialize method implemented by each record class. The rest of the classes are required for the RPC mechanism as described in Section 2.4.1.

5.3 Activities

For a better understanding of the ongoing process a sequence diagram is shown in Figure 5.4. It depicts the sequence of double clicking a node group. To simplify the diagram the user action at the beginning directly calls the LandscapeInteraction class. In fact, there are several steps before this call, but these are not relevant to the process. In the LandscapeInteraction several methods of the Usertracking class are called. In this specific case it is the trackNodeGroupDoubleClick method. In other cases the calls are made from the ApplicationInteraction class and for other mouse actions different methods are called, but these are only minor changes and the process stays basically the same. The several methods in the Usertracking class first execute the init method, in which the proxy class and the callback are instantiated. The proxy class is used for the communication between client and server. Due to the asynchronism of the RPC technique, the callback is used for passing the return value to the client-side. Then the record is created, in this case the NodeGroupOpenRecord is instantiated. Afterwards, the client passes the method call to the proxy class which passes the method call to the correct class on the server-side. This class, the UsertrackingServiceImpl, now serialises the record and writes it to the user tracking log file as already described. Finally, the return value which is actually only relevant in the case of failure, is sent back to the proxy class, which then calls the onSuccess or the onFailure method as appropriate.

Now we have an implementation fulfilling the requirements for tracking user actions in ExplorViz. The next step will be the evaluation which is the topic of the next chapter.

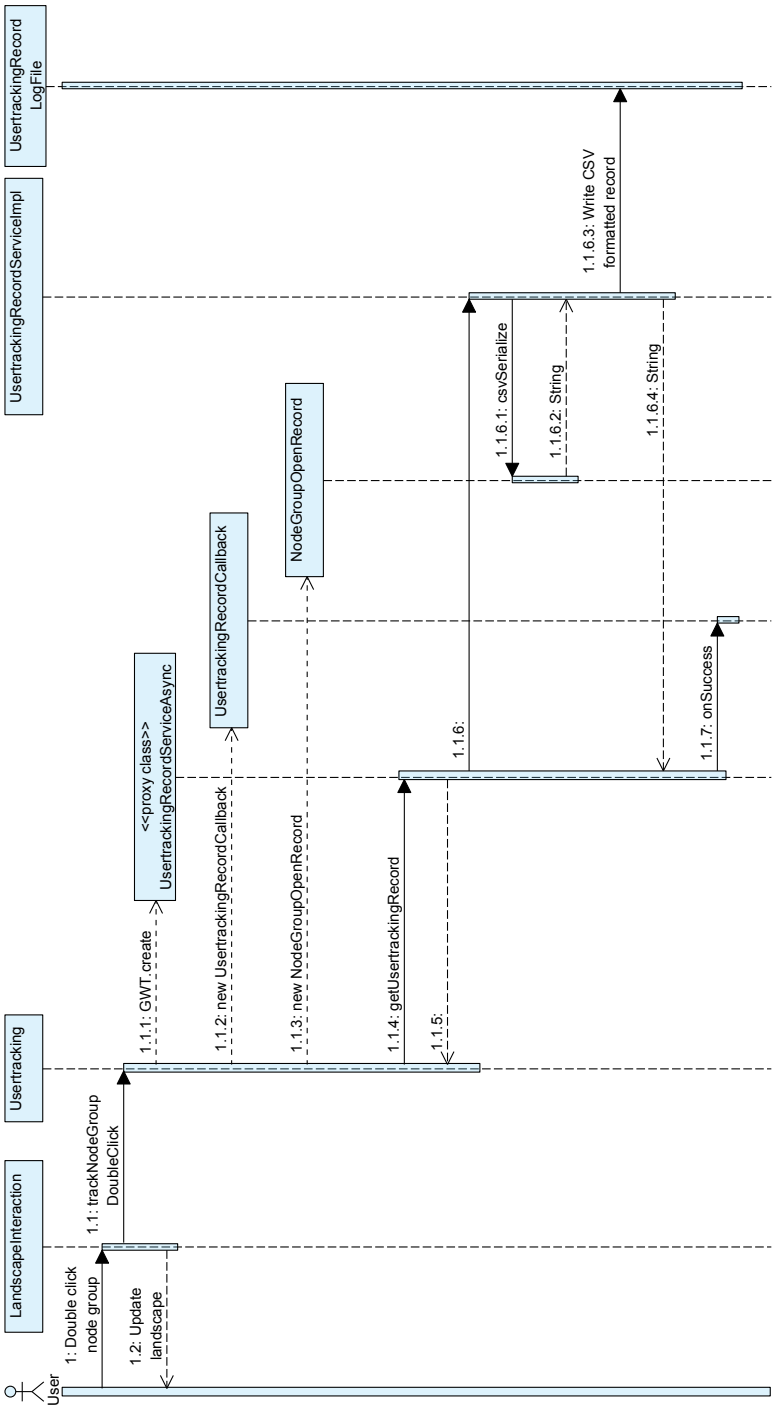


Figure 5.4. Sequence for double clicking a node group

Chapter 6

Evaluation of the Developed Tracking Tool

First, we will describe the quality criteria used for the evaluation of the developed tracking method. Afterwards, we will take a look at some scenarios and their resulting log files in Section 6.1.2 and Section 6.1.3. The concluding discussion on the results is conducted in Section 6.1.4.

6.1 Completeness Evaluation

6.1.1 Quality Criteria

The assessment criteria extend the criteria from Section 4.1. The criteria "Completeness", "Usability", and "Adaptability" are derived directly. "Correctness", "Standards", and "Analysability" are criteria for completing the evaluation and are again inspired by the CISQ. [Consortium for IT Software Quality 2012]

Completeness and Correctness The purpose of these criteria is to verify whether the implementation completely covers the requirements for tracking user actions in ExplorViz. The requirements include every action performed with the mouse. The landscape viewer and the system viewer also provide zooming actions with the mouse, but these actions shall not be tracked. Obviously, the resulting tracking data should be correct. This includes correct content and representation.

Usability This describes the usability from the user's view. In the best case the user experience is not influenced by the developed tracking method.

Adaptability After implementing the tracking method it is important that it is adaptable and expandable. It has to be possible to quickly add and change specific records or tracking methods.

Standards Resulting tracking data should be saved in a convenient format which should be highly compatible.

Analysability This criterion is important for further analysing and processing the tracking data and is connected to the chosen file format of the tracking data.

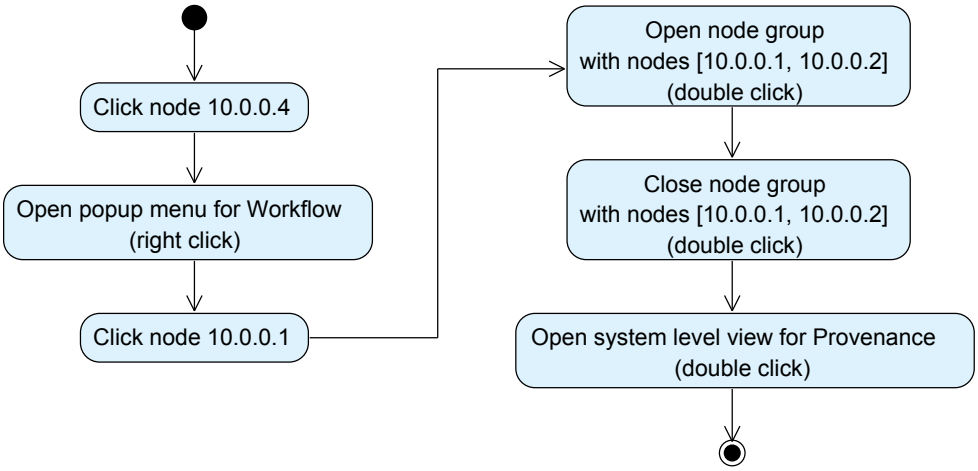


Figure 6.1. Activity diagram for scenario S1 in the landscape level view

For evaluating the completeness and correctness it is useful to investigate test cases. Comprehensive test cases were executed many times during and after the implementation. Three sample scenarios are outlined in the following. They can be easily reenacted with the project on the attached DVD in Appendix A.

6.1.2 Scenarios

The first scenario visualised in Figure 6.1 concentrates on actions in the landscape viewer. At the beginning, a click on the node with the IP 10.0.0.4 is performed. Through a right click, the popup menu of the application named “Workflow” is opened. After clicking another node with IP 10.0.0.1, a node group is opened by double clicking on it. The node group contains the nodes with IPs 10.0.0.1 and 10.0.0.2. Afterwards, the node group is closed again and an application named “Provenance” is double clicked. This leads to the system level view.

Figure 6.2 shows the scenario S2 for the system viewer. The system viewer contains different objects, but the actions performed do not differ from the actions in the landscape viewer. The first action is a right click on the component “jpetstore”. This action opens a popup menu. Afterwards, the component is closed and opened again. Subsequently, the component “persistence” is opened, then the component “sqlmapdao” is opened and finally, the popup menu for the class “AccountSqlMapDao” is opened. This popup menu shows the option for viewing the source code of the class which leads to the code viewer.

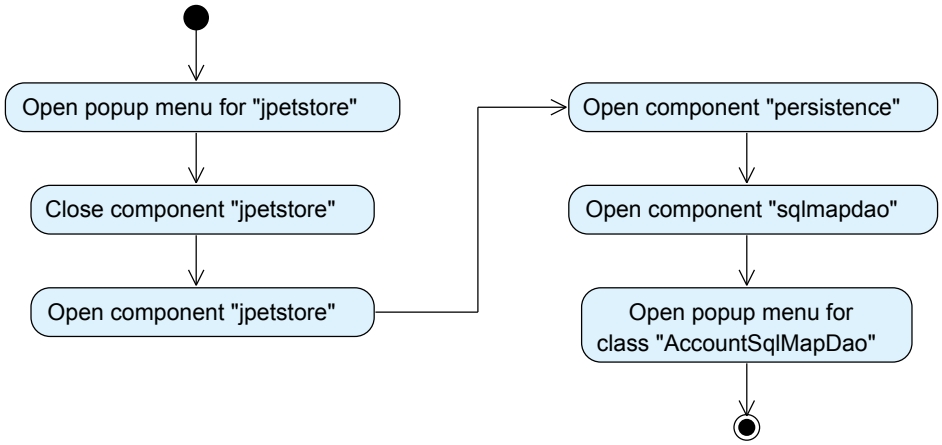


Figure 6.2. Activity diagram for scenario S2 in the system level view

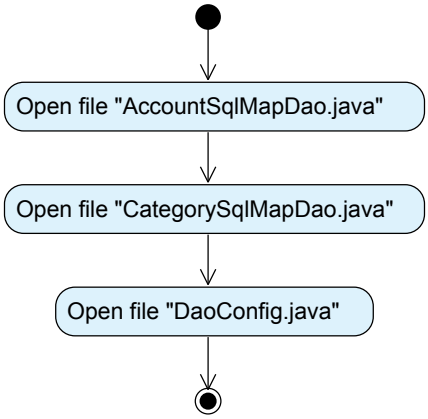


Figure 6.3. Activity diagram for scenario S3 in the code viewer

The last scenario S3 in Figure 6.3 concerns the code viewer. Although the user can perform expanding and folding actions on the tree hierarchy, only information about opened files are tracked in this view. Therefor, only three files are opened one after another during the scenario.

6.1.3 Results

When we now take a look at the resulting user tracking log file in Listing 6.1, the performed actions from scenario S1 can be recognised. At the beginning of each line is a time stamp which specifies the time in milliseconds. Following that, the class name of the created

```
1379514290635;NodeClickRecord;10.0.0.4;[Workflow-3,Provenance-4]
1379514316821;ApplicationOpenPopupMenuRecord;Workflow;3
1379514328773;NodeClickRecord;10.0.0.1;[Jira-0]
1379514333587;NodeGroupOpenRecord;[10.0.0.1,10.0.0.2]
1379514356923;NodeGroupCloseRecord;[10.0.0.1,10.0.0.2]
1379514383642;ApplicationOpenSystemLevelRecord;Provenance;4
```

Listing 6.1. Resulting log file after execution of scenario S1 in Figure 6.1

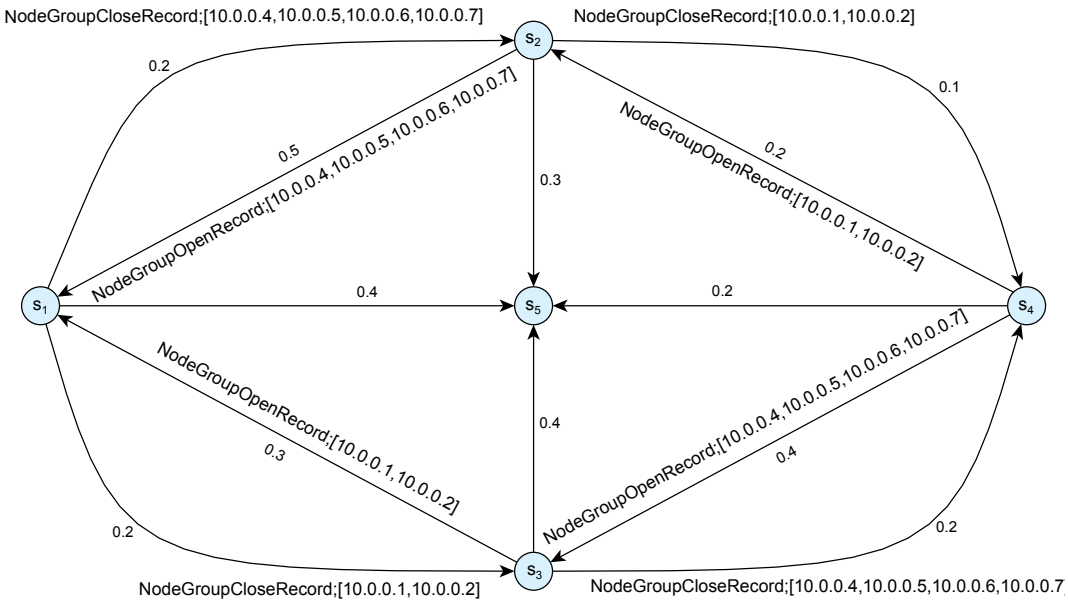


Figure 6.4. Possible Markov chain for the landscape viewer

record is displayed. This name encodes information about the type of the clicked object and exactly what action was performed. After this the properties of the record are given. In the first line a node with the IP 10.0.0.4 was clicked. Additionally we get the information which applications the node contains, namely "Workflow" with ID 3 and "Provenance" with ID 4. The application "Workflow" with ID 3 is the subject for the next action which opens a popup menu for it. The following lines reproduce the activity diagram in Figure 6.1 as expected. The last line describes the action of opening the system level view for the application with name "Provenance" and ID 4.

For investigating the scenarios, it may be of interest to connect the information of the performed actions. After collecting tracking data of multiple processes, it makes sense to visualise the data with a Markov chain. An example of a Markov chain for the landscape level view is given in Figure 6.4. It summarises the probabilities of changing the view by

```
1379514408153;ComponentOpenPopupMenuRecord;jpetstore
1379514466288;ComponentCloseRecord;jpetstore
1379514471539;ComponentOpenRecord;jpetstore
1379514488909;ComponentOpenRecord;persistence
1379514493043;ComponentOpenRecord;sqlmapdao
1379514515202;ClazzOpenPopupMenuRecord;AccountSqlMapDao
```

Listing 6.2. Resulting log file after execution of scenario S2 in Figure 6.2

```
1379514567194;CodeviewerOpenFileRecord;explorviz;JPetStore/com/ibatis/jpetstore/
    persistence/sqlmapdao/;AccountSqlMapDao.java
1379514612819;CodeviewerOpenFileRecord;explorviz;JPetStore/com/ibatis/jpetstore/
    persistence/sqlmapdao/;CategorySqlMapDao.java
1379523908729;CodeviewerOpenFileRecord;explorviz;JPetStore/com/ibatis/jpetstore/
    persistence/;DaoConfig.java
```

Listing 6.3. Resulting log file after execution of scenario S3 in Figure 6.3

either opening or closing node groups or by moving to a view of a different level. Other actions performed by the user have no influence on the number of executable actions, therefore they do not cause a transition to another state. The start state s_1 represents the initial landscape level view in which every node group is closed. There are two node groups which can be opened from here. This is done with the given probabilities in the graph and the state moves to s_2 or s_3. Continuing, the node group which is still closed can be opened as well or the already opened node group can be closed again. In the first case the transition moves forward to s_4, in the second case back to s_1. From here, we can close one of the node groups to move back to s_2 or s_3. It is possible to move to state s_5 from every state. It generalises the transition to the system level view which can be performed by double clicking any application. In fact, there should be a transition to a new state for every double clicked application. Self transitions caused by simple clicks on nodes or opening popup menus with a right click are not explicitly drawn as there is no need of drawing them as explained in Section 2.1.1.

Now we check the resulting log file in Listing 6.2. Beginning each line with a time stamp, the log file contains every performed action as expected and the correct record types are used for the different actions. At the end the `ClazzOpenPopupMenuRecord` represents the right click action performed on the class with name "AccountSqlMapDao". Since the opened popup menu provides an option for displaying the source code of the class, we use it for a transition from system viewer to code viewer.

The same actions as in the activity diagram of scenario S3 are presented in the log file given in Listing 6.3. Among the time stamp and the description of the record, information about project name, file path, and file name are contained.

6.1.4 Discussion

Completeness and Correctness The test cases of the last section verify completeness and correctness of the implementation. As expansion the navigation through the tree structure in the code viewer could be tracked.

Usability For the reason that tracking is only executed in the background, there is no impact on the usability for users. Plus, the log file can be accessed through a file explorer.

Adaptability The records can be easily complemented. The abstract class which has to be implemented by each record provides a time stamp and ensures that every record implements the `csvSerialize` method. A problem can be seen with the integration of tracking functions into core level concerning code. If tracking methods need to be added or changed, it takes more time to execute these changes than with aspect-oriented programming.

Standards The collected tracking data should be available in a format which is easy to read and also compatible to applications used for further analysis. The CSV format covers these requirements. It is an simple text format and is therefore highly compatible with many file viewers. Besides, the related tool Kieker uses the same format.

Analysability The CSV format is easily readable without dedicated analysis tools. Therefore, isolated information can be directly taken from the log file. For connecting information and providing a comprehensive analysis, however, the data has to be processed further. Markov chains or web graphs, which are described in Section 2.1, make the resulting data more analysable in terms of the probabilities of certain events. This aspect was already mentioned in Section 6.1.3.

6.1.5 Threats to Validity

For rating the degree of validity reached by the evaluation, we take a look at the possible sources of error. First of all, the depth of coverage is limited. Since the number of paths resulting from performing possible actions in different orders is infinite, not every path could be tested. However, the logging is stateless. This means, generating the records and writing them to the CSV file does not depend on the tracked actions before or afterwards.

Furthermore, the evaluation of the adaptability can only be seen as an estimate. Since we cannot foresee whether and to what degree the structure of ExplorViz will change, the dimensions of necessary adaptations are unpredictable. Nevertheless, the user tracking functionality is as modular as possible. The new classes are packaged in `usertracking`

packages both on the server and the client. Hence, potential changes concentrate on few packages and classes. Finally, if an instrumentation framework can be implemented instead of directly integrating tracking method calls into the business logic code, the interface between user tracking and core level concern will be defined more accurately.

The last point concerns the analysability. The processing of the tracking data is done manually so far. The goal of auto-generated visualisations will make it necessary to reevaluate the analysability of the tracking records, but the records are constructed following a pattern which can be easily parsed. As Kieker shows, the further processing and analysing is possible with the CSV format.

Chapter 7

Conclusions and Future Work

This last chapter concludes with related work in Section 7.1, an overall summary in Section 7.2, and future work in Section 7.3.

7.1 Related Work

Other tracking related tools are Google Analytics and Piwik, which are already described in Section 4.2 and Section 4.3. Compared to these tools the developed tracking method of this thesis is limited to the requirements of ExplorViz. It does not collect needless information such as location and software used by the users. There is also no need for an account and the tracking data is directly saved on the sever-side instead in an external application. One of the main differences is the coding language used. While Google Analytics and Piwik require using JavaScript, the implemented method in this thesis is written in Java for GWT.

Kieker is another related tool and was already introduced in Section 3.2.1. It provides many features and is therefore more complex than the implemented tracking method. Contrary to the method developed in this thesis, however, it employs method interception and provides further analysis for the tracked data.

7.2 Conclusions

The initial goal was the development of a tracking method for the web-based front end of ExplorViz. To that end, different tools and methods were evaluated in Chapter 4. The findings of this evaluation showed that current tools and methods are not fit to be used with GWT. Mostly they lack good documentation and are not compatible with current versions of GWT. As a consequence, the user tracking for ExplorViz was implemented with a manual tracking method combined with the GWT remote procedure call technique as described in Chapter 5. That way we have implemented a simple but working method. Besides, the logging method can be easily exchanged if an instrumentation framework provides support for the used technologies and the configuration of ExplorViz. The evaluation in Chapter 6 shows that the implemented tracking method meets the requirements.

7.3 Future Work

Future work includes an auto-generated visualisation of the logging data. The implementation developed in this thesis only produces log files, but Section 6.1.3 already shows the possibility of using Markov chains. This would make the information more readable and easier to understand. Furthermore, if looking for a very specific piece of information, a visualisation would be helpful for its quick retrieval. As introduced in Section 2.1.2 web graphs are another alternative for visualising the tracking data.

The rest of the future work lies in extending or improving the current implementation. The logging methods are written directly in the business logic code so far. This makes changes and extensions circuitous. Thus, the development of a working instrumentation framework would be useful. Some frameworks were already evaluated in Chapter 4. Either one of these frameworks could be extended and improved or a new method could be developed. Besides, frameworks with active development could offer a new release with more support for GWT in the future.

Finally, the user tracking itself could be extended. It is possible to navigate and zoom through landscape viewer and system viewer with keyboard and mouse. Moreover, the tree hierarchy can be expanded and folded. The hitherto existing tracking records are self-contained and provide a complete overview of the user's behaviour. Nevertheless, the navigation through the landscape viewer, the system viewer, and the tree hierarchy in the code viewer would complement the tracking data.

Bibliography

[Anttonen and Salminen 2011] M. Anttonen and A. Salminen. Building 3D WebGL Applications. Technical report 16. Tampere University of Technology Department of Software Systems, 2011. (Cited on page 7)

[Cantor 2012] D. Cantor. WebGL Beginner's Guide. Edited by A. Sheikh. Packt Publishing, 2012. (Cited on page 7)

[Chaganti 2007] P. Chaganti. Google Web Toolkit - GWT Java Ajax Programming: A Practical Guide to Google Web Toolkit for Creating AJAX Applications with Java. Edited by D. Chittar. Packt Publishing, 2007. (Cited on page 7)

[Chen et al. 2004] J. Chen, L. Sun, O. R. Zaïane, and R. Goebel. Visualizing and discovering web navigational patterns. In: *Proceedings of the Seventh International Workshop on the Web and Databases*. June 2004, pages 13–18. (Cited on pages 4, 5)

[Consortium for IT Software Quality 2012] Consortium for IT Software Quality. CISQ Specifications for Automated Quality Characteristic Measures. 2012. URL: http://it-cisq.org/wp-content/uploads/2012/09/CISQ-Specification-for-Automated-Quality-Characteristic-Measures.pdf (visited on 09/24/2013). (Cited on pages 15 and 27)

[Eclipse 2013] Eclipse. Xtend - Modernized Java: Documentation. 2013. URL: http://www.eclipse.org/xtend/documentation.html (visited on 09/09/2013). (Cited on page 8)

[Fittkau et al. 2013] F. Fittkau, J. Waller, C. Wulf, and W. Hasselbring. Live trace visualization for comprehending large software landscapes: the explorviz approach. In: *1st IEEE International Working Conference on Software Visualization (VISSOFT 2013)*. Sept. 2013. (Cited on page 11)

[Google 2013] Google. Google Analytics. 2013. URL: http://www.google.com/analytics/ (visited on 09/24/2013). (Cited on page 16)

[GWTProject 2013] GWTProject. GWT RPC Tutorial. 2013. URL: http://www.gwtproject.org/doc/latest/tutorial/RPC.html (visited on 09/09/2013). (Cited on page 8)

[Johnson 1997] R. B. Johnson. Examining the validity structure of qualitative research. *Education* 118.2 (1997), pages 282–292. (Cited on page 13)

[Kieker 2013] Kieker. Official Website. 2013. URL: http://kieker-monitoring.net/ (visited on 09/27/2013). (Cited on page 10)

[Lübbe 2012] K. Y. Lübbe. Improving a Transformation of Java Models to KDM. Bachelor's Thesis. Kiel University, Sept. 2012. (Cited on page 8)

[Munshi et al. 2008] A. Munshi, D. Ginsburg, and D. Shreiner. OpenGL ES 2.0 programming guide. Pearson Education, 2008. (Cited on page 7)

[Norris 1998] J. R. Norris. Markov Chains. Cambridge University Press, July 1998, pages 1–9. (Cited on page 3)

[Rohr et al. 2008] M. Rohr, A. van Hoorn, J. Matevska, N. Sommer, L. Stoever, S. Giesecke, and W. Hasselbring. Kieker: continuous monitoring and on demand visualization of java software behavior. In: *Proceedings of the IASTED International Conference on Software Engineering 2008 (SE'08)*. Edited by C. Pahl. Anaheim, CA, USA: ACTA Press, Feb. 2008, pages 80–85. (Cited on page 10)

[Schmücker 2011] N. Schmücker. Web tracking. SNET2 Seminar Paper. Berlin University of Technology. 2011. URL: http://www.snet.tu-berlin.de/fileadmin/fg220/courses/SS11/snet-project/web-tracking_schmuecker.pdf. (Cited on page 9)

[Seidelin 2011] J. Seidelin. HTML5 Games: Creating Fun with HTML5, CSS3, and WebGL. John Wiley and Sons, 2011. (Cited on page 7)

[Shadish et al. 2002] W. R. Shadish, T. D. Cook, and D. T. Campbell. Experimental and Quasi-Experimental Designs for Generalized Causal Inference. Houghton Mifflon Company, 2002. Chapter 1, 14. (Cited on pages 12, 13)

[Shreiner et al. 2009] D. Shreiner et al. OpenGL Programming Guide: The Official Guide to Learning OpenGL, Versions 3.0 and 3.1. 7th edition. Addison-Wesley Professional, 2009. (Cited on page 7)

[tutorialspoint 2013] tutorialspoint. GWT - RPC Communication. 2013. URL: http://www.tutorialspoint.com/gwt/gwt_rpc_communication.htm (visited on 09/09/2013). (Cited on page 8)

[Van Hoorn et al. 2009] A. van Hoorn, M. Rohr, W. Hasselbring, J. Waller, J. Ehlers, S. Frey, and D. Kieselhorst. Continuous Monitoring of Software Services: Design and Application of the Kieker Framework. Technical report TR-0921. Kiel University, Nov. 2009. (Cited on page 10)

[Van Hoorn et al. 2012] A. van Hoorn, J. Waller, and W. Hasselbring. Kieker: a framework for application performance monitoring and dynamic software analysis. In: *Proceedings of the 3rd joint ACM/SPEC International Conference on Performance Engineering (ICPE 2012)*. ACM, Apr. 2012, pages 247–248. (Cited on page 10)

[Wargolet 2011] S. Wargolet. Google Web Toolkit. Technical report 12. University of Wisconsin - Platterville Department of Computer Science and Software Engineering, 2011. (Cited on page 7)

[WebGL 2013] WebGL. WebGL puplic wiki. 2013. URL: http://www.khronos.org/webgl/wiki/Main_Page (visited on 09/09/2013). (Cited on page 7)

Appendix

Appendix A

ExplorViz DVD

Contents

/	The root directory contains this bachelor's thesis as pdf file.
/project_files/	Contains the project "ExplorViz" and every project dependency. The included readme file contains instructions on how to execute ExplorViz with the IDE Eclipse.
/project_files/ExplorViz/	Besides the packages and the class files edited for this thesis, this directory contains the user tracking file at war/UsertrackingLogFile.csv.

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Kiel,

_____